National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

A Visual Query Facility for Multimedia Databases

BY

Ghada El-Medani

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Fall 1995

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-06466-2

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Ghada El-Medani

TITLE OF THESIS: A Visual Query Facility for Multimedia Databases

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1995

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed) Ghada El-Medani

Ghada El-Medani
2 El-Saada Str. Osman's Bldgs. apt # 101
Roxy, Heliopolis
Cairo, Egypt

Date: June 26, 1995

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Visual Query Facility for Multimedia Databases** submitted by Ghada El-Medani in partial fulfillment of the requirements for the degree of Master of Science.

Dr. M. Tamer Özsu (Co-Supervisor)

Dr. Duane Szafron (Co-Supervisor)

Dr. Craig Montgomerie (External)

Dr. Ling Liu (Examiner)

Date: June 21, 1995

To my parents

Abstract

Multimedia databases store large data objects with complex spatial and temporal relationships. For users to easily and effectively access and make use of this large information space, there is a need for an interactive user interface to act as an intermediary between the database and the user. Many existing multimedia systems provide a browsing-based user interface for their users. However, with large information spaces, browsing does not provide an efficient mechanism for accessing the database. Thus, there is a need for a query facility which enables users to pose queries to the database, and, directly, retrieve information of interest.

This research determines the user interface requirements of multimedia information systems and provides a template solution. That is, it presents an easy and efficient way of accessing a multimedia database. This is achieved through a visual query facility which provides a rich visual query interface as well as a browsing facility. The visual query facility provides an interactive graphical interface, relieving users from having to type complicated queries which can be difficult. The system also provides users with the facility to customize system settings to suit their preferences and needs.

The design and implementation of the visual query facility is targeted towards a news-on-demand application which is a distributed multimedia news application that allows its users/subscribers to remotely access multimedia news articles inserted in a distributed database over a broadband ATM network. However, the same design and implementation principles can be applied to other distributed multimedia applications.

Acknowledgements

I would like to extend my thanks to both my supervisors Dr. Tamer Özsu and Dr. Duane Szafron for all their guidance, patience, and funding. This work could not have been completed if it was not for their direct involvement and feedback. They have made my year as enjoyable as it is fruitful. It was a great pleasure working and learing from them.

I am grateful to the members of my examining committee, Dr. Ling Liu and Dr. Craig Montgomerie for taking the time to read and comment on my thesis. Thanks go to Dr. Renee Elio who chaired my defense.

Thanks to members of the database laboratory for being a wonderful group to work with. Special thanks go to Chiradeep Vittal who was a good collegue and a very helpful partner in the multimedia project.

Many thanks go to my sister Sherine, my friends Ghada and Michael, and my fiance Maged for all their understanding. They have given me great support. Finally, I would like to thank my parents who have encouraged me and seen me through my years of study.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis Scope and Organization 3

2 Related Work **5**

3 The News-On-Demand Application **10**

3.1 Overview 10

3.2 Application Scenario 11

3.3 System Architecture 13

3.4 Requirements For the User Interface 14

4 System Design and Functionality **18**

4.1 Design Principles 18

4.1.1 Hypermedia 18

4.1.2 Query Facility 19

4.1.3 Input Modality 20

4.2 System Functionality 20

4.2.1 Viewing/Browsing Documents 21

4.2.2 Filtering Documents 21

4.2.3 Searching Documents 22

4.2.4 System Customization 22

4.3 Generalizing the Design 24

5 The User Interface **25**

5.1 News-on-demand User Interface 25

5.2 Generalizing the Interface 46

6 The Multimedia Type System **48**

6.1 Modeling of Monomedia Objects 49

6.1.1 Atomic Types 49

6.1.2 Storage Model for Text 49

6.2 Modeling of Document Structure 50

6.2.1 SGML Markup and Elements 50

6.2.2 Type System for Elements 50

6.3 Modeling of Presentation Information 52

7 Querying - Linking to the Database **54**

7.1 ObjectStore Queries 54

7.2 The Type System and Queries 58

7.3 Classes of Queries 60

7.3.1 Queries on Articles' Attributes 60

7.3.2 Queries on User Profiles 61

7.3.3 Search Queries 62

7.4 Implications of Using ObjectStore 66

8 Implementation Issues **68**

8.1 The Smalltalk User Interface 68

8.2 The C++ Query Agent 71

8.3 Alternatives for Implementation 73

9 Conclusion and Future Work **75**

9.1 Conclusion 75

9.2 Unimplemented Features 76

9.3 Future Enhancements 76

Bibliography 79

A User Profile Class Definitions 84

B Valid Command Strings 88

C Glossary 95

List of Figures

1 Distributed System Architecture 13

2 Multimedia News Startup Window 26
3 MMnews Launcher 27
4 Launcher Icons 27
5 Document Filter 30
6 Media Settings 32
7 MMnews Global Search 34
8 MMnews Global Search (Text Submenu) 35
9 Search Result 36
10 Document List 37
11 Document Abstract 39
12 Document History 40
13 Document View 42
14 Document Icons 43
15 Style Sheet Window 44
16 Document Editorial Information 45
17 Viewing an Image 47

18 Atomic Types Hierarchy 49
19 First-Level Element Type Hierarchy 51
20 Type Hierarchy for Other Text Elements 51
21 Type Hierarchy for Structured Text Elements 52

22 Type Hierarchy for HyTime Elements 53

23 Visual Query Components 69

24 Smalltalk Main Modules 70

25 C++ Main Modules 72

Chapter 1

Introduction

1.1 Motivation

One of the major reasons for the recent popularity of multimedia information systems is the opportunities they provide for easier and more effective human-computer interaction (HCI). The significance of multimedia systems lie in enhancing the communication between computers and human users [Ada93]. Traditionally, text was the primary medium for interacting with computer systems, thus limiting users to a restricted framework for interaction. Today, multimedia systems incorporate and integrate information from diverse media sources, such as text, graphics, images, audio and video, presenting the users with various channels for communication and information delivery, thus allowing for a broader and richer means for interaction. This increases system usability as humans communicate more effectively through various channels [BD92].

The media sources, incorporated in multimedia documents, involve large data objects with complex spatial and temporal relationships. This results in a large body of information, with a need for an effective means for managing and accessing it. Traditional file systems fall short when trying to fulfill this requirement as they leave to the user/application the responsibility of properly formatting files for the multi-

media objects as well as the management of the data itself. The problem becomes more complicated in the case of multi-user multimedia systems where concurrency control, security, and application independence are very important issues. Multimedia systems can benefit from database management system (DBMS) services such as data abstraction, high-level access through query languages, application independence, controlled multi-user access (concurrency control), fault tolerance, and access control (security). Multimedia systems are generally distributed, requiring multiple servers for their storage requirements. Thus, distributed DBMS technology can be used to efficiently and transparently manage data distribution. Object-Oriented techniques should be used to provide encapsulation, abstraction, and extensible type systems. Therefore, we proclaim the use of an object-oriented multimedia database at the heart of multimedia information systems [VÖSEM94].

A multimedia database efficiently stores multimedia objects and models the relationships between them. For a user to easily and effectively access and make use of this large information space, there is a need for an interactive user interface to act as an intermediary between the database and the user. Thus, a considerable portion of achieving the usability and effectiveness desired from multimedia systems lies in being able to provide users with easy-to-use effective interfaces for accessing information.

Many multimedia information systems provide a browsing-based user interface for users. Perhaps, the most popular interfaces of this type are Mosaic are Netscape for accessing the World Wide Web. However, with large information spaces, browsing does not provide an efficient mechanism for accessing the database. This research focuses on adding a querying capability that enables users to retrieve information of interest. Users can still browse through the retrieved information in the same manner as in the browsing-based systems. The novel aspects of this approach are the following:

- Combining querying and browsing to provide a richer interface.

- Relieving users from having to type complicated queries (which can be difficult) by using a visual querying interface.

1.2 Thesis Scope and Organization

This thesis determines the user interface requirements of multimedia information systems and provides an easy and efficient way of accessing a multimedia database. This is achieved through a visual query facility which provides a rich visual interface for querying the database as well as a browsing facility. The visual query facility is not intended to provide a complete user interface that adequately covers all the needed functionality for multimedia information systems. Instead, its main purpose is to provide the users with an easy and efficient way of accessing the multimedia database. However, the thesis looks at the requirements of a user interface for such systems even though the full interface is not implemented. The following aspects are central to this research:

- Studying the user interface requirements for multimedia information systems.
- Designing a visual query interface to relieve users from typing complex database queries.
- Tightly integrating the interface with the multimedia database.
- Separating the logical document content from presentation to provide a completely customizable system.

The design and implementation of the visual query facility is targeted towards a news-on-demand application. However, the same design and implementation principles can be applied to other distributed multimedia applications.

The news-on-demand application is described in detail in Chapter 3. It follows a distributed client/server model. The client application environment includes the visual querying facility (the focus of this thesis), the synchronization routines [LG94], a

quality-of-service negotiation component [VBD+93], and the ObjectStore [LLOW91] database client [Vit95]. The clients are connected to a set of servers over a broadband ATM network. The servers are responsible for the storage and management of multimedia documents. This research is part of the Broadband networking services project supported by the Canadian Institute of Telecommunications Research (CITR). Several other universities are involved in this project: University of Waterloo, University of Ottawa, Université de Montréal, INRS Telecommunications, and University of British Columbia.

The rest of the thesis is organized as follows. Chapter 2 provides a discussion of the approaches and systems dealing with accessing multimedia information spaces. Chapter 3 describes the target application, news-on-demand, and the constraints it imposes on the user interface and querying facility. Chapter 4 discusses the functions provided by the system and the design principles followed to fulfill the system functional requirements. Chapter 5 discusses the user interface in detail. Chapter 6 describes the document type system. Chapter 7 describes the underlying querying of the multimedia database. Chapter 8 discusses important aspects and choices regarding the implementation. Chapter 9 summarizes the thesis, describing unimplemented features and possible enhancements and future developments.

Chapter 2

Related Work

The design and specification of interfaces for information systems have gained the attention of many researchers in a variety of fields including query languages, hypertext systems, and user interfaces. This is due to the fact that the power of information systems can only be realized when tools are provided for effective access to the stored information. In this chapter, query languages are first discussed, followed by hypertext systems. Finally, systems that combine both to achieve better functionality are highlighted.

Query languages and database access tools provide users with the facility to access stored information. Structural Query Languate (SQL), being the most popular query language in the relational database world, provides a variety of functions to manipulate data [Eme89]. For simple data sets, relational databases and SQL provide adequate functionality, but as the need arose to store and manipulate complex data such as multimedia information, they fell short of providing the necessary functionality. Object-oriented database management systems (OODBMS) provide capabilities for modeling complex data. Some OODBMS query languages are SQL extensions such as O2 [D+91] and ONTOS [Ont91], while others are new languages such as GEMSTONE [MS87] and ORION [K+90]. ObjectStore provides a query facility by defining C++ types and methods [OHMS92]. Some OODBMSs come with an ad-

vanced graphical user interface for designing, browsing, and modifying the database schema [ADE93]. However, their query facilities are text-based. To use these query facilities, users have to learn the query language syntax and the data model. This poses a problem to system users who do not possess a deep knowledge of computer science. One approach to overcome this problem is to define graphical query languages that allow users to query information in the database graphically, rather than by formal query languages. Some of these systems are:

- DOODLE (*Draw an Object-Oriented Database LanguagE*) is a visual language which allows users to query object-oriented databases using arbitary pictures. DOODLE visual programs use the same semantics as F-logic [Cru92].

- Image databases are often coupled with query by visual example capabilities which enable the user to retrieve images by providing rough sketches or diagrams. Examples of such systems are described in [HK92, KFS90].

In addition to visual query languages, research work is being done on query languages supporting multimedia issues, namely the temporal and spatial aspect of information retrieval. In [DG92], EVA, a query language with multimedia support, is described.

Hypertext systems promise to deliver information to users in new and unconventional ways. As early as the 1950s, Doug Engelbart was working on hypertext concepts and office automated systems, providing frameworks for hyperlinks and collaborative work. In 1967 and 1968, Andries Van Dam developed the first two working hypertext systems known as: Hypertext Editing System and FRESS. Nine years later, in 1978, the first hypermedia system, Aspen Movie Map, was developed by the MIT Architecture Machine Group [Nie90a]. Today, many researchers are developing hypertext and hypermedia systems. Literature provides reviews of a variety of systems from different application domains which employ the concepts of navigating information spaces and following links to related information. Many multimedia information

systems provide elaborate hypertext graphical user interfaces with limited search capabilities. Perhaps, the most popular interfaces of this type are Mosaic and Netscape for accessing the World Wide Web. Akscyn [AMY88] describes KMS (Knowledge Management System), a distributed hypermedia system that supports organization-wide collaboration. In addition to its browsing facility, KMS allows its users to search for text strings in the hypertext nodes. In [Wil91], a hypermedia system for on-line technical documentation is presented. It provides its users with a browsing facility as well as a limited keyword search facility. Examples of other hypermedia systems are presented in [GT94, HKR+92, Pla91, BD92]. Along the same lines, a model for hypertext-based information retrieval is presented in [Luc90].

The complexity of current multimedia applications raise more challenges to meet the ease and efficiency requirement of user accessibility. Halasz [Hal88] indicates that providing search and query facilities is the number one issue in hypermedia systems of the future. Research has been conducted to develop systems which combine the strengths of query languages and hypertext navigation. No standard specification for the design and implementation of such systems has yet been reached.

One approach is to define systems that integrate a functional query language with navigational facilities. The users can retrieve information by following links (navigation) or by querying the database using the query language. Examples of such systems include:

- Frei [FS91] describes a system which supports storage, retrieval as well as browsing of multimedia information. These functions are achieved by providing hyperlinks between related information as well as a searching facility. Queries are handled using a functional database query language (FQL*) which is extended to include elementary similarity functions to allow for imprecise queries.

- MMIS is a multimedia information system developed at the University of Manchester [GOC+92, O'D93]. It currently handles the storage and retrieval of text

and raster images. Users can store media instances on the system's server using different editors. After storing a medium instance, a module must be activated to interpret that instance producing semantic nets which are subsequently used for querying. Prolog queries can then be entered and processed against the stored semantic nets.

Another research direction is using a graphical query language which manipulates hypertext concepts. Andonoff [ACMZ91, ACMZ92] describes OHQL (Object Hypertext Query Language) which uses hypertext concepts to query object-oriented databases. The database is presented to the user in the shape of a hypergraph with nodes representing classes and links representing inheritance and domain constraints. Users formulate queries by graphically selecting nodes and links on the hypergraph.

Other systems integrate query facilities together with hypertext capabilities within the same graphical user interface to provide users with a uniform way of accessing multimedia information. Individual systems vary in the facilities they provide.

- Keim [KKL91] describes a system where natural language captions are used to describe the contents of multimedia data. During retrieval, users express their queries via a graphical user interface using natural language to describe the query constraints. The query is represented graphically by a set of boxes (representing simple constraints or subqueries) and connections between them (to represent operators). The system allows users to define their queries in an incremental fashion. The system then translates user queries into a formal query language (SQL) to execute on the underlying entity-relationship data model.

- In [Fer94], the KIM (Knowledge-based Information System) query system is described. KIM provides users with homogeneous access to distributed, heterogeneous, multimedia databases. The system provides both iconic and diagrammatic (entity-relationship) interfaces to the database. Users select an entity or icon and define filters in the retrieval process by defining the attributes of

interest and the selection conditions. Queries are associated with icons for later usage or update. The system supports incremental development of queries.

Chapter 3

The News-On-Demand Application

3.1 Overview

News-on-demand is an application that provides its subscribers/users with access to multimedia news documents that are inserted into a distributed database. Multimedia documents are inserted into the database by news providers. Examples of news providers are television networks, newspapers, magazines, and wire services. Once inserted in the database, the multimedia news documents cannot be updated. The subscribers, at different client sites, access the news database, via a broadband network, to retrieve and read articles according to their own interest. The cost of this service to the users includes the cost of the information content as well as the transmission and retrieval costs. The users can affect the cost they have to pay by specifying quality-of-service parameters for the system's operations.

In this context, a multimedia document is defined *"as a structured collection of pieces of information related to a particular subject"* [VÖSEM94]. The information units can include any media type such as text, image, audio and video. They can also include combinations of different media, such as video synchronized with audio and text captions. Documents often have links to other information (whether complete

documents or components of documents). Examples of such links are more news coverage, background information, and expert analysis. The structure of the document consists of two entities: the *logical structure* and the *presentation structure*. The logical structure is concerned with the logical organization of the document components. For instance, logically, a news article is made up of a headline, a number of paragraphs, figures, and so on. The presentation structure is concerned with the layout of the document; in other words, how the document is actually displayed. Examples of parameters of interest to presentation are font typeface, font sizes, the number of columns in which text is displayed, etc.

The news-on-demand application suggests several issues of interest to the querying facility and user interface:

- There is a need for a standard representation for the documents inserted in the database. The standard chosen is SGML/HyTime [VÕSEM94]. By following a standard, the system provides a uniform document representation that facilitates document entry by news providers and access by users.

- Different subscribers/users are likely to customize their system view due to personal preferences and/or hardware constraints (e.g., absence of a color graphics display). The user interface should allow such customization.

- There is a need for separating the logical structure of documents from the presentation structure when storing documents in the database. This separation allows for a more customizable system where presentation issues can be decided upon by the users without affecting the documents' logical structure.

3.2 Application Scenario

The interaction between the user and the multimedia news database is achieved by posing queries to the database and receiving results. The visual querying interface is responsible for translating the user's requests into actual queries that are executed

on the database. The results obtained from the database are presented to the user via the interface as well. The following example scenario illustrates the interaction between the user and the system:

> Typically, the user wishes to view news articles on a specific subject. Using the visual querying facility (as will be explained in more detail in Chapter 5), the user specifies the subject he/she is interested in. The system returns a list of headlines of articles that meet the user's request. The user can then view the abstracts of any of the returned articles. The user selects a particular article to view. The system retrieves the article and displays it. The user can read the article, view images, listen to attached sound recordings, play video, or follow links to other information of interest.

The above scenario is only an example of a possible interaction between the user and the system. Users can perform many more types of queries. Examples of these queries are:

- Return documents whose media source a particular news provider.
- Return documents with a certain location, date, and/or keywords.
- Return documents written by a certain author.
- Return documents which contain text, but not video.
- Return documents with a particular text string within the text of the article.
- Return documents whose images, audio and/or video contain certain keywords.
- Return the images which contain certain keywords.
- Return the abstracts and/or paragraphs which contain a particular text string.

Currently, the query facility provides extensive support for searching document attributes (headline, location, date, author, media source, etc) and text elements

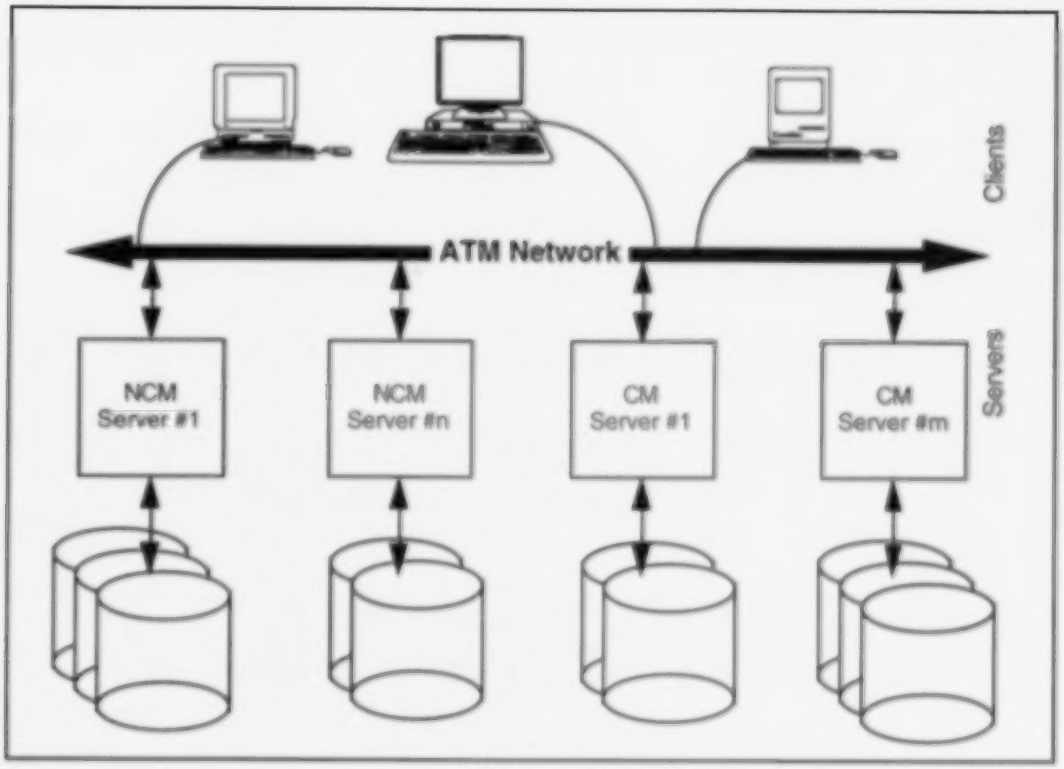

Figure 1: Distributed System Architecture

(abstract, sections, paragraphs, quotes, etc). Keyword searches are provided for other media types: images, audio and video. Apart from queries on documents and/or document components, users can pose queries regarding user settings of the system and/or presentation settings. Queries for retrieving meta-information about continuous media are another category of queries to the database.

3.3 System Architecture

The news-on-demand application follows a distributed multiple server multiple client model (Figure 1). The clients are connected to a set of servers over a broadband ATM network. The servers are responsible for the storage and management of multimedia documents. There are two kinds of servers, depending on the type of media they manage and store: the continuous media servers (CM servers), and the non-continuous

media servers (NCM servers). Continuous media refers to media with real-time constraints, such as audio and video. Non-continuous media include text and still images which do not have the real-time constraints of audio and video. Currently, the server database integrates the non-continuous media servers, but not the continuous media servers. Instead, the database stores meta-information about files stored on the CM servers. The meta-information is used, by the client, to locate the file and then it is accessed directly from the CM file server. Several system modules reside on the client side. The client application environment includes the visual querying facility, the synchronization routines [LG94], the quality-of-service negotiation component [VBD+93], and the database client. The distribution of the data among the servers is transparent to the clients since all accesses to the servers are done through the client DBMS and the other system modules.

The testbed environment for the system consists of client machines which are IBM RS6000/360's with 128 Mbytes of memory. The server machines are RS6000/360's with 64 Mbytes of memory. The servers and clients are connected via a Newbridge ATM switch [1]. A commercial object-oriented database management system, ObjectStore, is used as the database engine.

3.4 Requirements For the User Interface

The news-on-demand application suggests several requirements for the user interface of such a system. This section first discusses the general requirements imposed on multimedia user interface design. Then it focuses on the requirements specifically emerging from the news-on-demand application.

Since multimedia systems have a lot of potential in delivering information to the

[1] This testbed environment is not exactly replicated at the University of Alberta where the client and server machines are connected by an Ethernet network

users, special attention should be given to the actual usabilility of these systems. A system's usability is determined by how easily and effectively the users can communicate with the system to achieve a desired task. Usability parameters for most systems include ease of use, efficiency, ease of remembering, and pleasantness [Nie90a]. Some general design requirements for usable multimedia user interfaces are [Hay93]:

- Simplicity: A multimedia user interface should be simple to use, requiring only a minimal cognitive load from the user. Overwhelming the user with complex icons, too many choices, various color patterns (as might be the trend in some of the current multimedia interfaces) may not always help. Instead, such interfaces will distract the user from the real purpose of the application.

- Consistency: Providing a consistent design throughout the system makes it easier and more predictable for the user to follow. This reduces the efforts that the user puts in trying to follow the interface and allows the user to concentrate on the actual information presented.

- Engagement: Engagement is determined by the degree to which the user can participate, affect and control the actions of the system. "*Multimedia should invite the user to participate*" [Hay93]. Interactive interfaces provide added value by providing the user with the means to interact with the system.

- Depth: A multimedia interface should encourage users to explore the system to a greater depth by making it easy to do so. However, it should not force the user to understand the system to any greater depth than the user wishes to explore.

- Fun: Multimedia user interfaces should be fun. This will encourage more people to use the system and each person to use the system in more ways. Note that the goal is not to encourage users to waste time using the system. The goal is to encourage them to use the system in novel, but productive ways.

The above general design requirements apply to the news-on-demand application as well as all multimedia applications. For any multimedia application, the user interface should be simple, consistent, inviting for users to participate and explore, and pleasant to use. However, there are more specific requirements of news-on-demand users. Users of news-on-demand are expected to use the system to achieve several tasks related to the news. These tasks include analysis of financial and political situations which can affect planning, regular follow-up of news, and access to background and reference news material. The general high-level functions required of the system can be summarized as follows:

- Viewing/Browsing information: Users should be able to view multimedia documents, read text, look at images, play video, listen to audio, and follow links to related information.

- Searching for information: In addition to the browsing facility, users should be able to search the multimedia database with a variety of criteria such as date, author(s), category, location, keywords, etc. Searches on document content should also be provided. The system should provide a fast and easy way for searching.

- Customizing the system: Users should be able to define and modify their own settings of the system. Settings can include: documents layout, window specifications, quality-of-service parameters and others.

- Other functions: These include, but are not limited to, allowing users to add their own annotations to news documents, providing users with additional navigational aids such as subject indexes and a history of the visited documents.

These requirements point to a customizable and easily extensible interface which combines the browsing capability (found in most existing multimedia interfaces) with a querying capability (lacking in many of the same interfaces). Furthermore, the querying capability should be a visual one that merges seamlessly (is consistent)

with the browsing facility and satisfies the other general usability requirements of a multimedia user interface.

Chapter 4

System Design and Functionality

4.1 Design Principles

In the previous chapter, the various design goals and requirements for multimedia user interfaces in general and for the news-on-demand application in particular are examined. According to these requirements, the design principles chosen can categorized into three major points:

- Hypermedia
- Query facility
- Input modality

4.1.1 Hypermedia

Hypertext/Hypermedia provides the user with a non-sequential means of freely browsing information according to his/her needs by following links from one information unit to another [Nie90a]. Links are maintained between related units of information, providing users with several options to follow when reading a unit of information, thus enriching the user interaction with the system. Traditionally, hypertext networks link units of text together. In case of hypermedia systems, information units

can contain text, graphics, images, audio and video. Hypertext/Hypermedia systems can also provide navigational aids such as subject indexes, maps, tours, backtracking facilities, graphical representation of the hypertext network, history of visited nodes, etc. to facilitate browsing information. Note that not all multimedia systems use hypermedia links [BD92] unless it is required as part of the application functionality.

In the news-on-demand application, multimedia news documents often have logical links to other related information such as background information, more news coverage, follow-ups, and expert analysis. Therefore, a hypermedia interface is a good design choice as it provides the news readers with an easy and efficient way of accessing and browsing related information.

4.1.2 Query Facility

A hypertext/hypermedia interface to a multimedia system may not always be sufficient to provide all of the accessing mechanisms the user needs to obtain information from a database. In many applications, such as news-on-demand, users need to search for specific information based on partial knowledge. This must be achieved more simply and quickly than is possible through the browsing facilities of hypermedia. Moreover, as the information increases in quantity and complexity, the browsing facility of hypermedia becomes more and more inadequate. According to studies [Nie91, Nie90b] of the usability of hypertext systems, users have often reported that they become disoriented while navigating through hypertext systems and fail to reach points of interest. This phenomenon was reported even when using the most popular commercial hypertext systems. For the news-on-demand application, a querying facility that allows users to search and retrieve information directly from the database is a good design choice. The user interface should provide an easy way for performing queries and searches, as well as examining the results. A typical scenario is for the user to first filter the list of documents in the database to include only relevant news articles. The user then performs a search for documents and document components that are of interest for specific topics and then browses this limited set. The design

must support this incremental process.

4.1.3 Input Modality

A *"good"* user interface should provide the user with appropriate interaction modes, depending on the application and the types of input needed from the user. The modes of interaction can be categorized into three major categories [BD92]:

- direct manipulation of graphical objects on the screen;
- the use of natural language;
- the use of formal languages.

In the case of the news-on-demand application, as well as other multimedia applications, a graphical user interface with direct manipulation techniques is sufficient to deal with user input. Typically, users need to click on icons to follow links, choose options from menus and lists, type text, etc. A natural language interface would be a great facility if provided with the graphical interface. However, this is beyond the scope of this research. The need for using formal languages (in this case, a query language) has been eliminated by designing the visual query interface to make the system more usable, especially by novice users. In other words, the visual query interface is used as a front-end to the ObjectStore database management system's query constructs.

4.2 System Functionality

The system's functionality for the visual query facility falls under four main categories: viewing and browsing documents, filtering documents, searching documents, and system customization.

4.2.1 Viewing/Browsing Documents

The system allows the users to view and browse multimedia documents. Users can view a document's abstract before fetching the whole document from the server. This helps users decide the relevance of a document before retrieving it. Once a document is viewed, users can read its text, look at images, play video, listen to audio, and follow links to related information (although following hyperlinks is not implemented in the current version of the system, it is still recognized as a main function of the system). Users can also view the document's editorial information, such as: author(s), location, media source, date, etc.

4.2.2 Filtering Documents

Filtering is a search on document attributes. It is used to specify the scope of documents that the user views/browses. The result of a filtering operation is a set of documents whose attributes match the ones specified by the user. A filtering operation is done by defining a filter object and applying it to the set of documents stored in the database. Filters can be based on news category, keyword(s), location, headline, media source, author(s), and/or date. The user can also filter documents depending on the type of media they include, so users can choose only to view documents which contain certain media types and not others. This feature is used to answer queries such as: *"Return documents with text and no audio"*. Filter objects are stored in the database in the user profile. Users can identify one filter to be the default filter which is used during system startup, but filters can be applied to document lists anytime. The system also defines a set of filters to provide short cuts for users. These include filtering news documents published today (using system date) and filtering documents on news category.

4.2.3 Searching Documents

The system provides users with the capability of searching documents' content for specific information (in addition to searching on document attributes which is provided through filtering). Users specify the text to search for (optionally using Boolean combinations of text strings). For completeness, both conjunctive and disjunctive normal forms are supported as search expressions (explained in more detail in Chapter 5). The media types to be searched and the scope of the search are also specified by the user. The scope of the search can be all the documents in the database, the documents in a list, only the current document, or documents directly linked to the current document. As mentioned before, the user specifies the media types to be searched. In case of text, the user can select all or any number of text elements to be searched. These include headline, abstracts, sections, paragraphs, quotes, etc. Documents' editorial information (part of the document attributes) such as category, location, keyword(s), source, and author(s), can also be searched. In case of other media types (images, audio, and video), a keyword search is performed. We are ultimately interested in enhanced searching of images, audio, and video to provide content-based searching and access techniques. The search facility allows the user to query the database and locate specific information directly as needed. This provides an easier and more efficient way of accessing data than using the navigation facility where exploration can be very time-consuming.

The search facility returns a list of objects that match the query. The objects returned can be text paragraphs (or any text element), images, video clips, audio streams, or complete documents. The types of objects returns are specified by the user during the search (they correspond to the media types to search for).

4.2.4 System Customization

Another important feature of the interface is allowing the users to customize the system. Customization ranges over a variety of system parameters, such as defining

default settings for system startup, as well as presentation preferences.

Style Sheets

Style sheets store user preferences with regards to presentation of documents. Information stored in style sheets include defining the presentation of anchors (which are the departure points to hyperlinks) within a document, attaching a font format (such as italics, bold, reversed video, etc) to an emphasized text, and selecting a primary font for the document. Users can define different style sheets to render different presentations of the same document. The concept of separating the document's logical structure from the presentation structure makes this possible and easy to handle by the system. Style sheets are stored in the database in the user profile. Users identify one style sheet to be the default style sheet for displaying documents. However, users can apply style sheets to documents at any time.

Media Settings

Users can also define other presentation settings (apart from those in the style sheets) to customize the system. These include defining immediate playing of a video when the icon is clicked versus opening up a control panel (similar to a VCR) to allow explicit activation of the video. Users can also define whether they want to open windows for all of the media types contained in a document once the document is viewed or whether they should be displayed explicitly when clicking on a specific icon. Also, when following links, users can either choose to open every link destination in a separate window or in the same window. These settings are called media settings as they deal with the presentation of media types in the system. They are also saved as part of the user profile. There is only one set of media settings per user (in contrast to possibly several filters and style sheets) as users are likely to want to keep a uniform setting for presenting media irrespective of which document they are viewing.

User Profiles

User profiles contain all the user preferences. In other words, they are the central objects for system customization. Every user of the system has a user profile which stores his/her preferences. User profiles contain a list of filters, a list of style sheets, the default system filter, the default style sheet, and the media settings. User profiles are identified by user name. When a new user connects to the system, a user profile (with default settings) is automatically created for him/her. Users can update their profiles any time.

4.3 Generalizing the Design

The above system design and functionality, although discussed in the framework of the news-on-demand application, is extendable and applicable to other distributed multimedia information systems. A design which combines hypermedia browsing capabilities with a rich visual query facility in one graphical user interface provides adequate functionality for multimedia information systems. Allowing users to customize the system according to their preferences is also a desirable design feature in all software systems. Therefore, applications, where users need to search and browse information stored in a large multimedia database, are good candidates for such a design. Futhermore, these applications also require the same high-level system functionality described earlier for news-on-demand; namely: viewing/browsing documents, filtering documents, searching, and system customization. Although details of each function may vary depending on the specific application targeted, the higher level functions are still the same. For example, in most multimedia applications, filtering documents is a desirable feature. However, the attributes on which documents can be filtered will differ from one application to the other.

Examples of applications that can make use of this design are distant teaching applications, online museums, and multimedia library systems.

Chapter 5

The User Interface

5.1 News-on-demand User Interface

This section describes the user interface for the news-on-demand multimedia information system (also called MMnews in the interface windows). This discussion covers the full functionality of the user interface; a few of these features were not implemented in the current version.

The user interface for the news-on-demand multimedia information system provides a number of fundamental functions:

- initiate a quality-of-service negotiation;
- start a filtering operation in the database;
- perform a search;
- customize the system by defining media settings, style sheets, and filters;
- retrieve and display a document;
- display a list of visited documents.

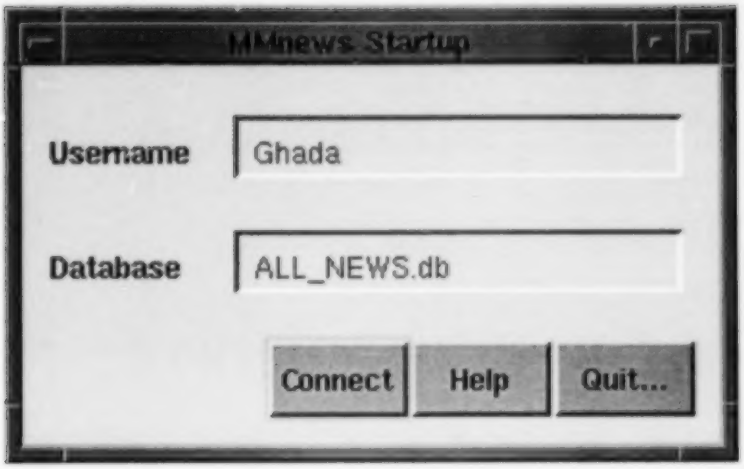

Figure 2: Multimedia News Startup Window

System Startup

When starting a multimedia news session, the startup window (Figure 2) is displayed, prompting the user to enter his/her username and the news database file the user wishes to connect to. A user can only connect to one news database file during a session. However, different files can be used in different sessions. Once the user chooses **Connect** to connect to the database and start the session, the system proceeds to fetch the user's profile. The username acts as an ID for fetching the user profile. In case of new users, the system automatically creates a new profile, with default settings, for the user in the specified database. The system then displays the MMnews launcher.

Application Launcher

The MMnews launcher is the gateway to the news-on-demand application. Through the launcher, the user can get to all the other services and functions provided by the system. The launcher (Figure 3) provides a set of menus as well as some action buttons. For an explanation of the launcher icons, refer to Figure 4. The launcher menus provide users with a variety of services.

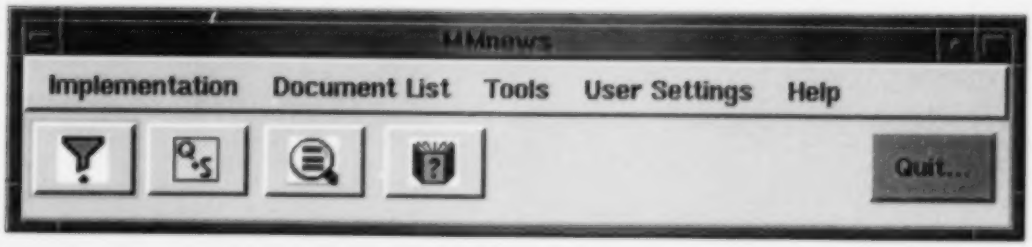

Figure 3: MMnews Launcher

Figure 4: Launcher Icons

Implementation Menu

Implementation menus are available in all the MMnews windows. They provide debugging capabilities to the system implementors.

Show ObjectStore query	Displays the ObjectStore queries that correspond to the various functions performed in that window. Queries are discussed in more detail in Chapter 7.
Inspect	Inspects the Smalltalk code of that particular window.

Document List Menu

All Documents	Displays a list of document titles of all the documents in the database.
Filtered Documents	Displays a list of document titles of documents filtered on the default filter specified in the user profile.
Empty	Opens a Document List window without any documents.

Tools Menu

Category Index	Displays a submenu of all the news categories in the system. By selecting any item of the submenu, a document list window containing the documents belonging to that category is displayed.
History	Gives a history of all the documents visited in this session. (Not implemented in this version)
Document Hotlist	Opens the user's document hotlist. The user can proceed by viewing any document in that list, etc. (Not implemented in this version)
News Today	Displays a list of document titles of news articles published today.

User Settings Menu

Filter	Displays a submenu of all the filters in the user profile. Users can edit any of these filters or create new ones.
QofS	Displays a submenu of all the quality-of-service parameters in the user profile. Quality-of-service is not linked to the query facility in this version.
Media Settings	Opens the media settings window which allows the users to define presentation preferences.

Help Menu

About MMnews	Displays a general message about the application.
About Launcher	Displays help on the functions of the MMnews launcher window.
Online Documentation	Provides online help for all the system with browsing capabilities. (Not implemented in this version)

Filter Window

As has been mentioned before, filtering is a search on document attributes. Its purpose is to reduce the scope of the documents the user sees, thus facilitating further searching and browsing of this limited set. A filtering operation is done by defining a filter object in the database and applying it to the documents in the database to get a reduced subset. Users can create and edit filters by means of the **Filter** Window (Figure 5). The Filter window allows users to specify the attributes they wish to search for. These include: keyword(s), location, category, source, headline, authors, time period, and media content. In case of locations, category, and source, the user is provided with a list of the items available in the database to reduce errors. The user only needs to define the attributes he/she is interested in searching for; the others can be left empty. Users can specify a filter to be the default by checking the **Save**

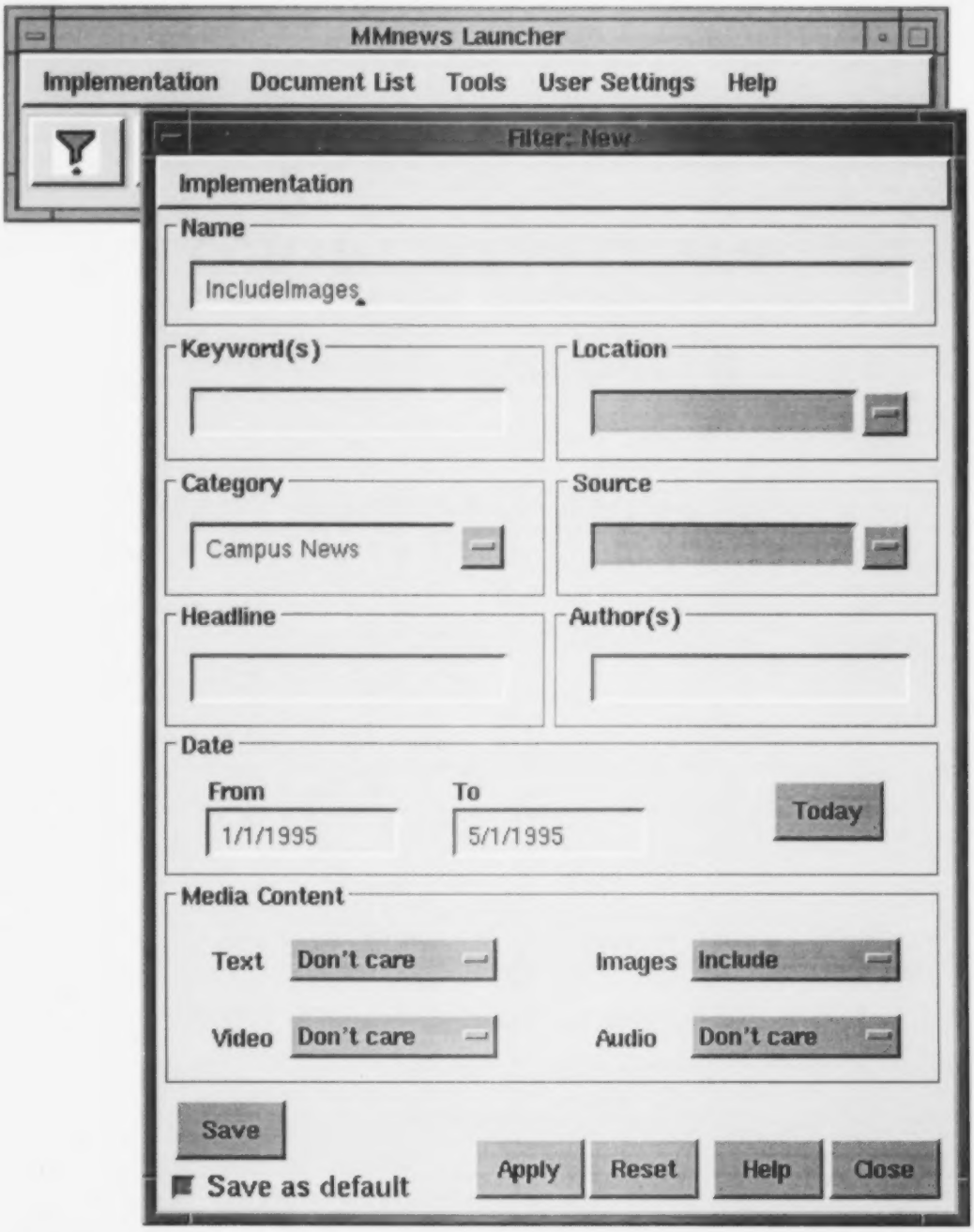

Figure 5: Document Filter

As Default checkbox. Every filter must have a name which is used as an ID for accessing the filter in the user profile.

In Figure 5, the user creates a filter to retrieve news articles that were published in the first five months of 1995 whose category is "Campus News" and that include images.

Media Settings Window

The **Media Settings** window (Figure 6) allows the user to specify some presentation settings concerned with the display of the various media types in the system. Users select their preferences by simply checking the corresponding checkboxes. Once the user modifies the media settings, the new settings are applied to the system immediately. For example, if, at any point of a session, the user chooses to view floating media (images, audio, and video) immediately when fetching the document, the system will, subsequently, display all the images, audio, and video included in a document once this document is fetched. The user can go back and re-modify the media settings anytime he/she wishes. Media settings can also be saved in the user profile to become the system default.

Search Windows

Users can perform searches on documents' content for specific information through the **Search** windows. Searches can be performed from the Application Launcher, from a document list, or from within a document. The **Search** window (Figure 7) allows users to specify the text string to search for, optionally using Boolean combinations of strings. The algebraic operators provided by the system are **AND** and **OR**. **NOT** is not provided because, in news-on-demand, the usual scenario is to search for information of interest to users, rather than the exclusion of information. For that reason, **NOT** was not provided. However, incorporating that facility is straightforward. The system supports both conjunctive and disjunctive normal forms for search expressions:

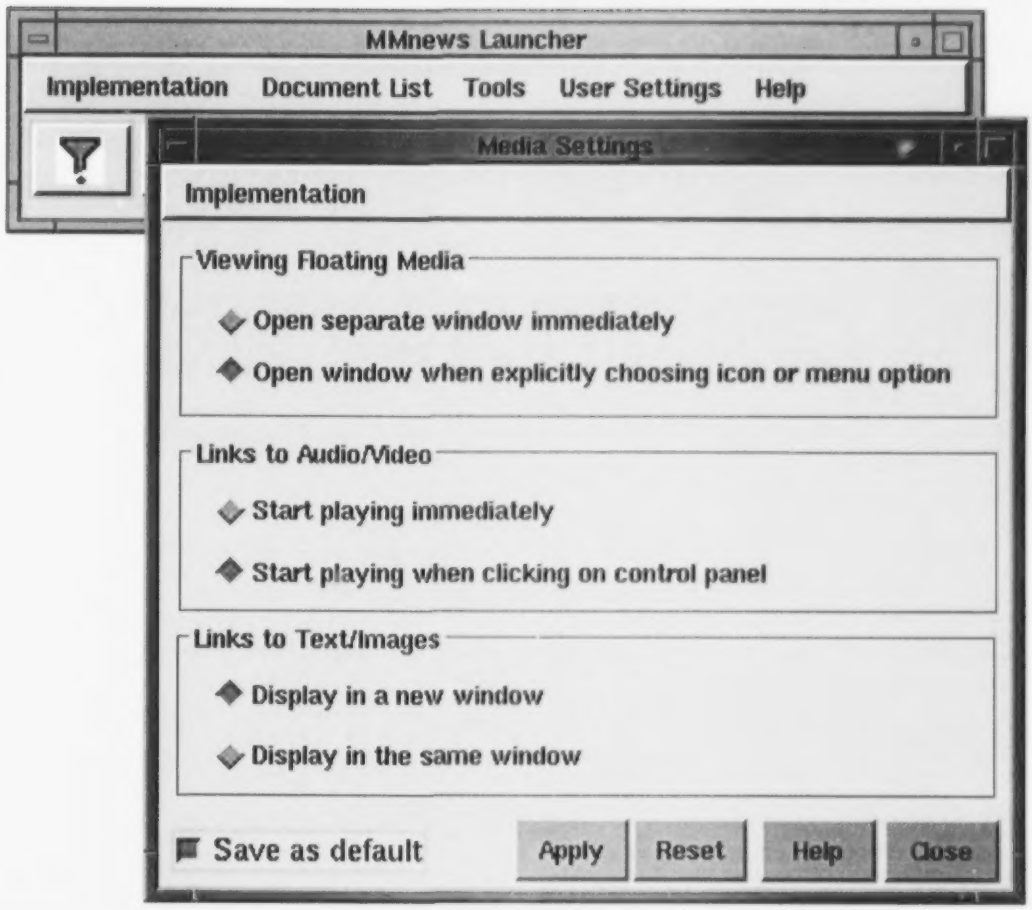

Figure 6: Media Settings

- (*SearchTerm1* **OR** *SearchTerm2*) **AND** (*SearchTerm3* **OR** *SearchTerm4*)
- (*SearchTerm1* **AND** *SearchTerm2*) **OR** (*SearchTerm3* **AND** *SearchTerm4*)

where the *SearchTerms* can be any textual substring surrounded by quotes (spaces can be included in substrings). The algebraic operators (**AND**s and **OR**s) are inserted when pressing the corresponding action buttons. Brackets are automatically provided by the system once an expression is built. The **Search String** is used to build sub-queries and then add them to the complete **Query** string. Of course, users can still build simpler queries consisting of one search term or two. Thus, the system provides the facility for complicated as well as simple query expressions. The user also specifies the media to be searched by checking the corresponding checkboxes for **Text**, **Images**, **Video**, and **Audio**; at least one of the media types must be checked. The text elements to search for should be specified by checking menu items in the **Text** submenu (Figure 8). Text elements include abstracts, headlines, paragraphs, sections, quotes, etc. The scope of the search can be all the documents in the database, the documents in a list, the current document only, or the documents directly linked to the current document. However, in case of the global search (initiated from the **Application Launcher** - Figure 7), the scope is fixed to all the documents in the database.

The search facility returns a list of objects that match the query (Figure 9). The objects returned can be text paragraphs, images, audio streams, video clips, etc or complete documents. The type of objects returned is dependent on the media searched, and retrieved types specified (whether **Documents** or **Components** or both). For example, if the user checked **Video** as the media searched and **Components** for retrieval, the search will return only video objects, but not the complete documents containing them. However, if the user also specified **Documents**, the complete documents that contain video components will also be returned.

In Figures 7 - 9, the user is searching for the words "Waterloo" and "Edmonton" in headlines, abstracts, images (figure captions), video, and audio (textutal descriptions). Accordingly the search returns the matching items depending on the **Retrieve**

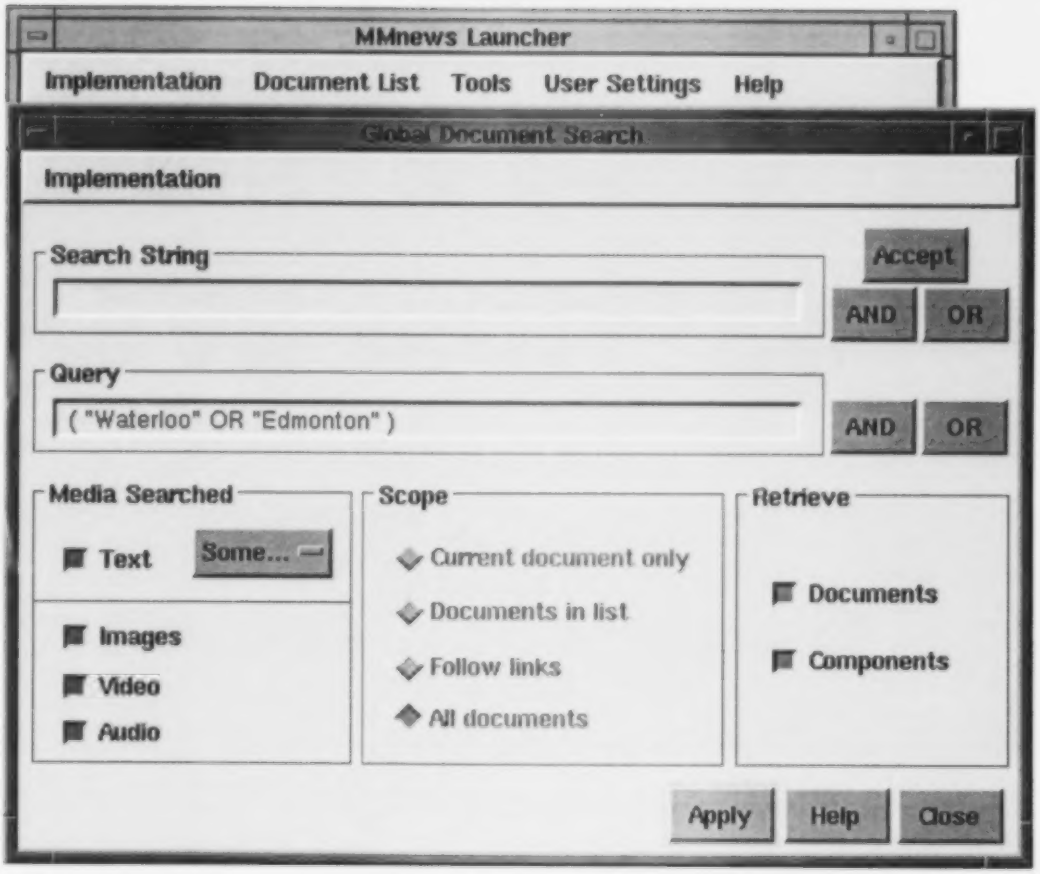

Figure 7: MMnews Global Search

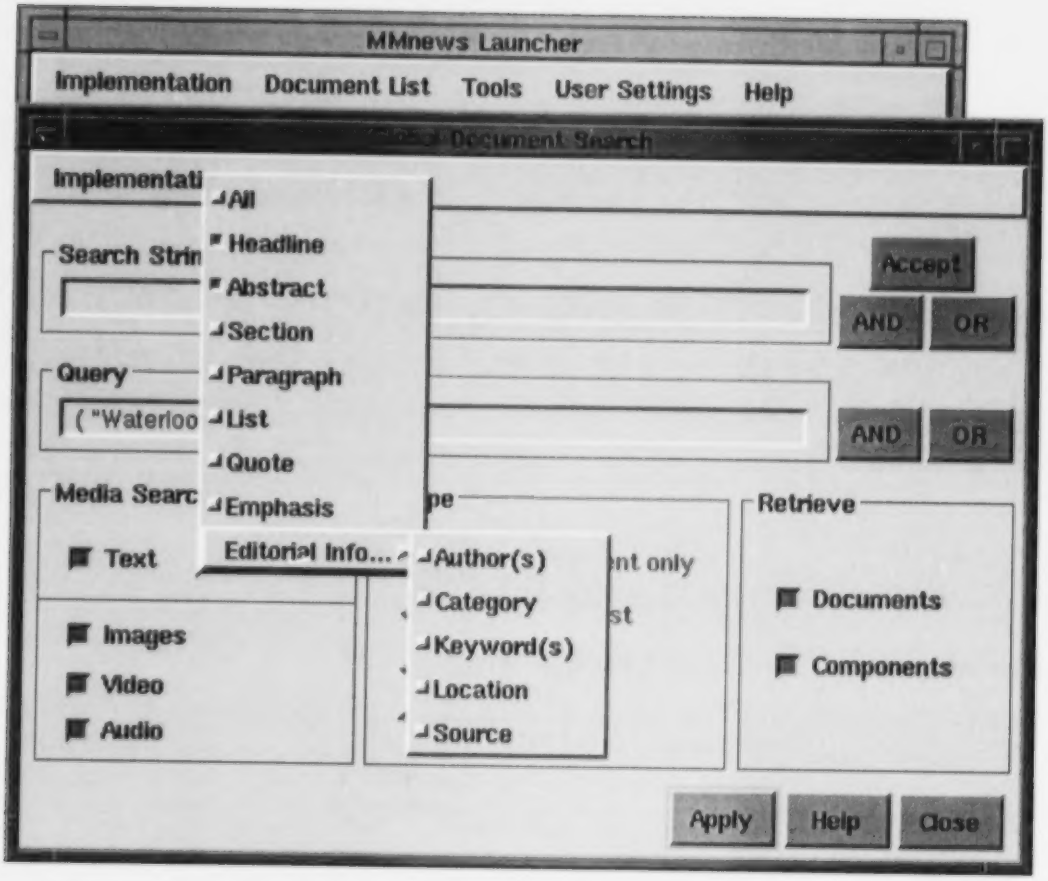

Figure 8: MMnews Global Search (Text Submenu)

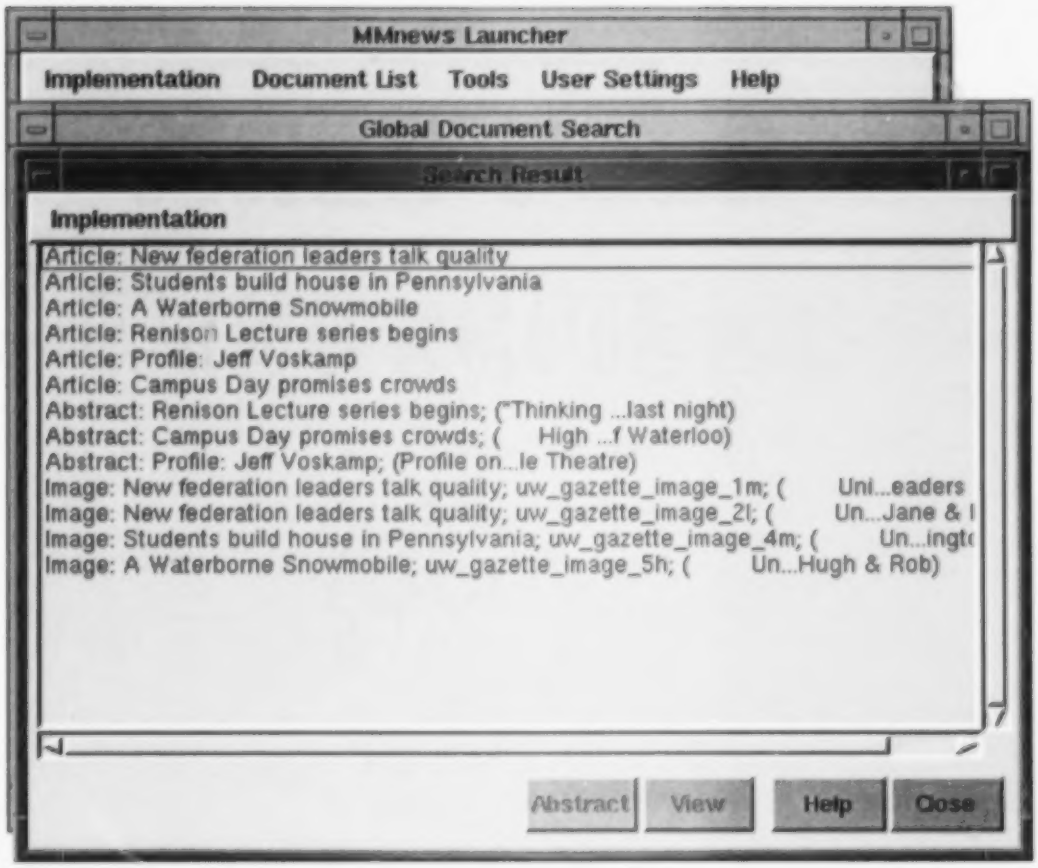
MMnews Launcher
Implementation
Document List
Tools
User Settings
Help
Global Document Search
Search Result
Implementation
Article: New federation leaders talk quality
Article: Students build house in Pennsylvania
Article: A Waterborne Snowmobile
Article: Renison Lecture series begins
Article: Profile: Jeff Voskamp
Article: Campus Day promises crowds
Abstract: Renison Lecture series begins; ("Thinking ...last night)
Abstract: Campus Day promises crowds; (High ...f Waterloo)
Abstract: Profile: Jeff Voskamp; (Profile on...le Theatre)
Image: New federation leaders talk quality; uw_gazette_image_1m; (Uni...eaders
Image: New federation leaders talk quality; uw_gazette_image_2l; (Un...Jane & I
Image: Students build house in Pennsylvania; uw_gazette_image_4m; (Un...ingt
Image: A Waterborne Snowmobile; uw_gazette_image_5h; (Un...Hugh & Rob)
Abstract
View
Help
Close

Figure 9: Search Result

Figure 10: Document List

options. In this case, articles, abstracts, and images are returned because the user specified both **Documents** and **Components** for retrieval.

Document List Window

The **Document List** window displays a list of document titles. This list can contain all the documents in the database or documents matching a certain criteria (a specific filter, news today, etc). Figure 10 depicts a **Document List** window showing documents filtered using the `IncludeImages` filter (Figure 5). Users can perform several actions from the **Document List** window:

Search	Allows the user to perform a search on the documents in the list, the selected document, or all the documents in the database. The search window and the search result are the same as those depicted in Figures 7 - 9.
Abstract	Displays the abstract paragraph of the selected document in the list (Figure 11). Viewing a document abstract helps users decide the relevance of the document before retrieving it.
View	Fetches and displays the selected document (Figure 13). The document is displayed according to the default media settings and style sheet.
History	Displays a list of all the previously visited documents from that window (Figure 12). The users can redisplay the abstracts or the complete documents again. This facility helps users easily locate information which was previously viewed.
Help	Displays help on the functions of the Document List window.

Filter Menu

Displays a submenu of all the filters in the user profile. Users can edit any of these filters or create new ones. Filters can be applied to the document list to change the scope of the documents listed.

QofS Menu

Displays a submenu of all the quality-of-service parameters in the user profile. Quality-of-service is not linked to the query facility in this version.

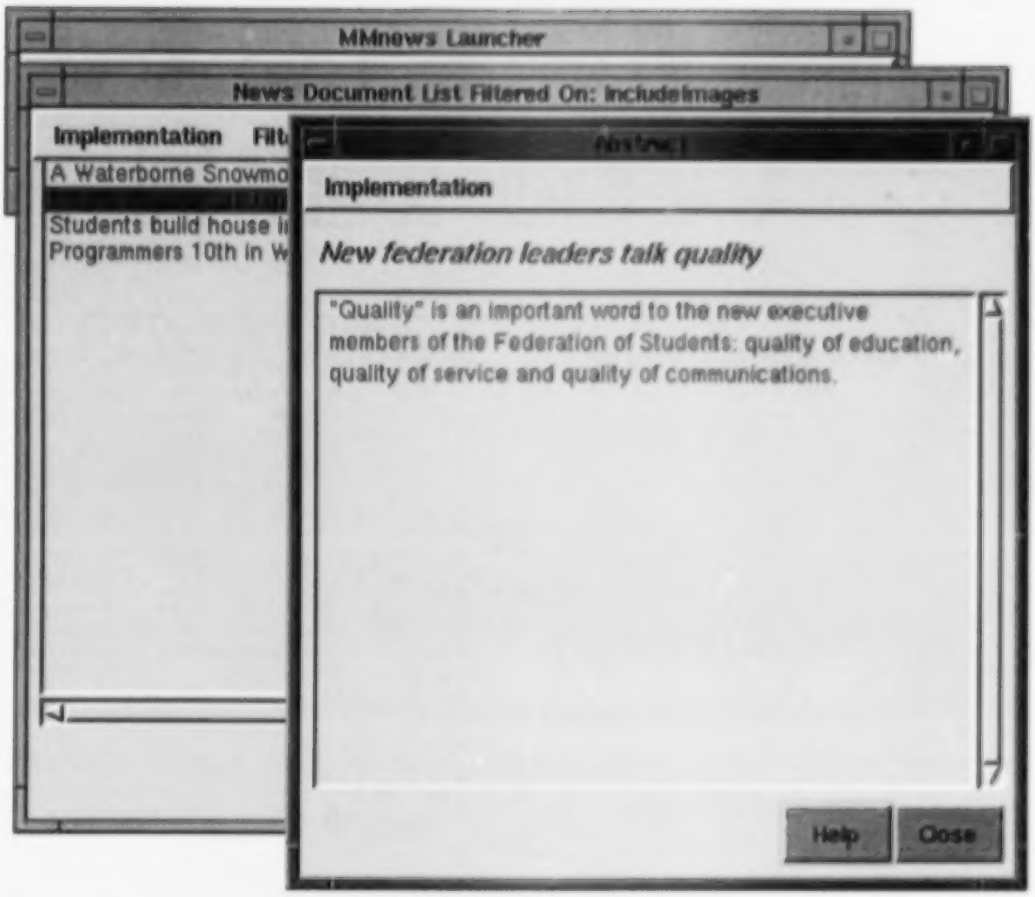
MMnews Launcher
News Document List Filtered On: includeImages
Implementation
A Waterborne Snowmo
Students build house i
Programmers 10th in W
Abstract
Implementation
New federation leaders talk quality
"Quality" is an important word to the new executive members of the Federation of Students: quality of education, quality of service and quality of communications.
Help
Close

Figure 11: Document Abstract

Figure 12: Document History

Document View Window

The **Document View** window (Figure 13) displays the different components of a document. It provides a set of menus and action buttons that allows users to perform several functions. For explanations of the different icons, refer to Figure 14. The text portion of the document, if any, is displayed in the window's text view. Users can perform a search from within a document by selecting the *Search* button. *History* and *Help* are also provided. In addition, the menus provide the following functions:

Style Sheet Menu

Displays a submenu of all the style sheets in the user profile. Users can edit any of these style sheets or create new ones. Style sheets can be applied to the document view to alter the presentation of the document. The **Style Sheet** window (Figure 15) will be opened when any of the style sheets in the menu is selected. The user can then specify the anchor presentation (underlined, borderd, iconified, or plain), the font format (bold, italics, or underline) attached to the emphasis elements, and the primary font (Times, Courier, or Helvetica) used for the document. These settings as well as the options given for each setting can be extended to allow for more presentation control.

Annotate Menu

The Annotate menu allows the users to *add*, *delete*, and *edit* their own annotations on the news articles. This feature is not implemented in this version.

View Menu

Editorial — Displays editorial information about the current document (Figure 16). Editorial information includes the article's headline, author(s), location, category, media source, date, and keyword(s).

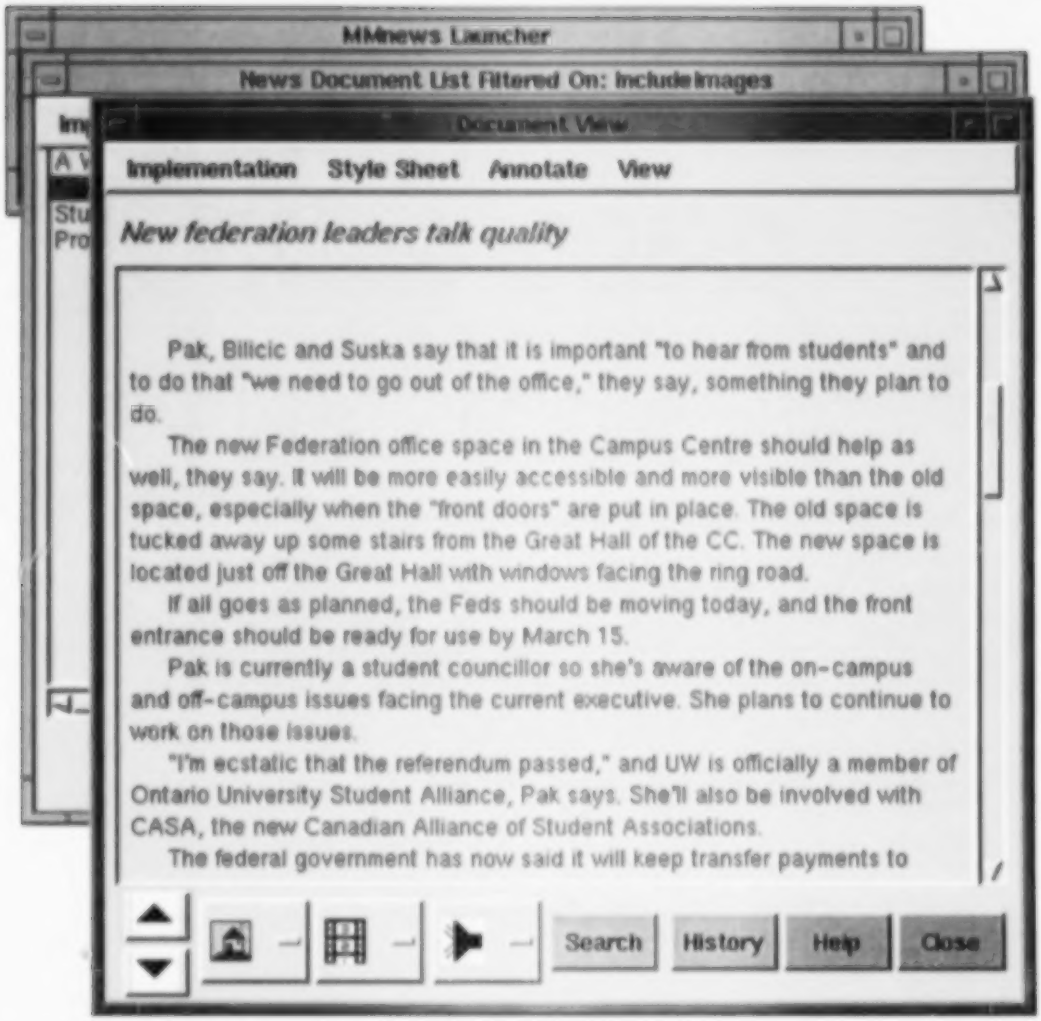
MMnews Launcher
News Document List Filtered On: includeImages
Document View
Implementation
Style Sheet
Annotate
View
New federation leaders talk quality
Pak, Bilicic and Suska say that it is important "to hear from students" and to do that "we need to go out of the office," they say, something they plan to do.
The new Federation office space in the Campus Centre should help as well, they say. It will be more easily accessible and more visible than the old space, especially when the "front doors" are put in place. The old space is tucked away up some stairs from the Great Hall of the CC. The new space is located just off the Great Hall with windows facing the ring road.
If all goes as planned, the Feds should be moving today, and the front entrance should be ready for use by March 15.
Pak is currently a student councillor so she's aware of the on-campus and off-campus issues facing the current executive. She plans to continue to work on those issues.
"I'm ecstatic that the referendum passed," and UW is officially a member of Ontario University Student Alliance, Pak says. She'll also be involved with CASA, the new Canadian Alliance of Student Associations.
The federal government has now said it will keep transfer payments to
Search
History
Help
Close

Figure 13: Document View

Figure 14: Document Icons

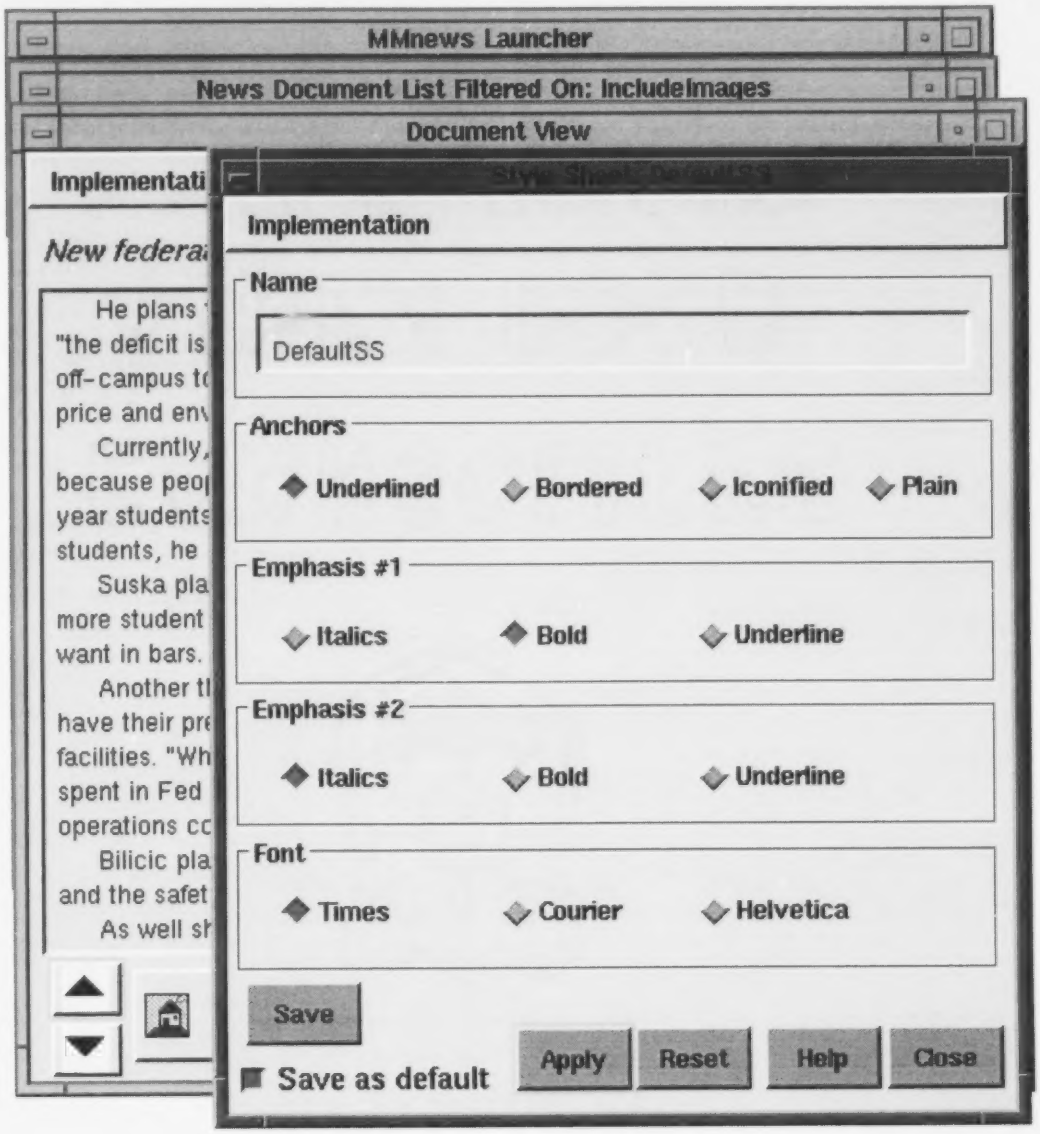

Figure 15: Style Sheet Window

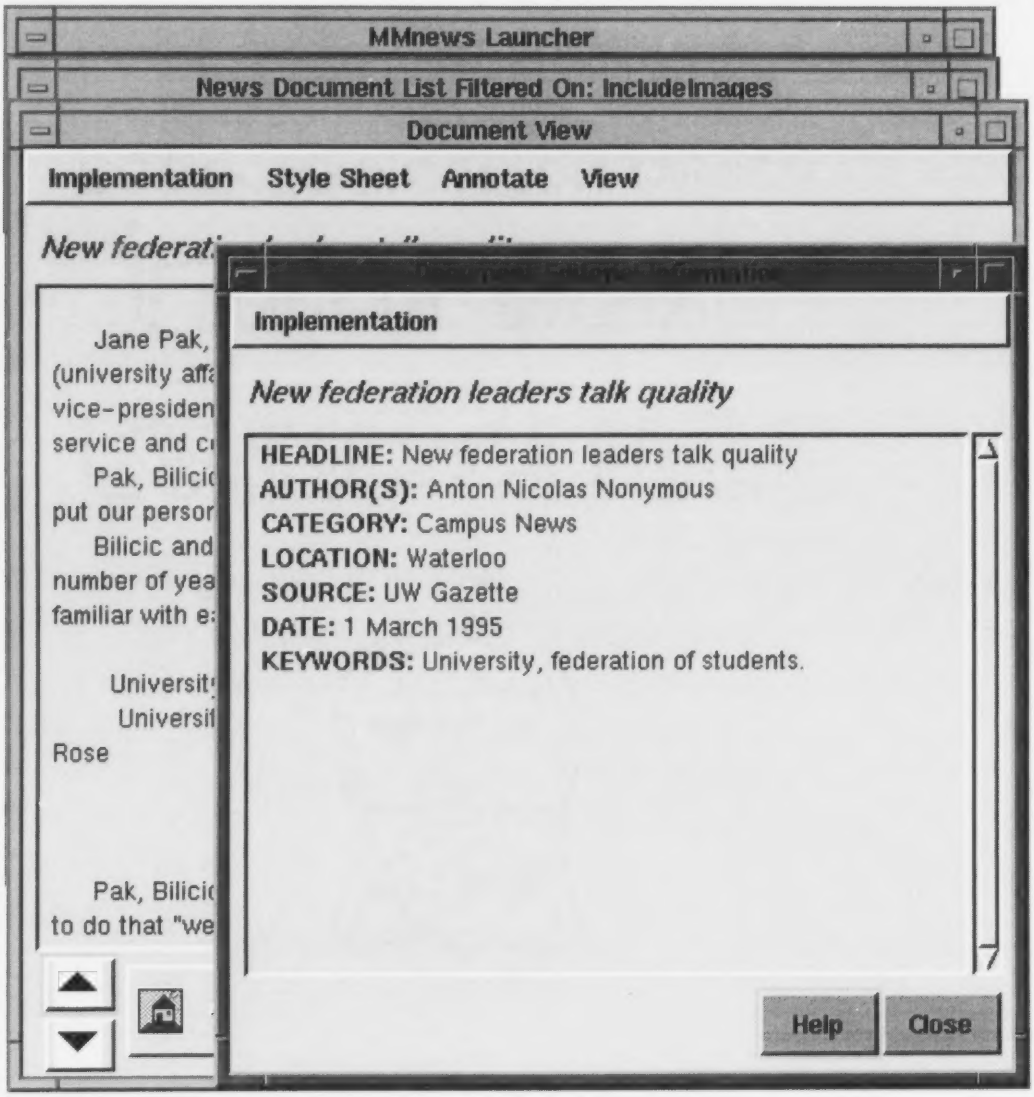

Figure 16: Document Editorial Information

Images	Displays a submenu of the names of images (if any) included in this article. The user can display any of these images by clicking on its name in the images submenu. Figure 17 shows a displayed image from the current document.
Video	Displays a submenu of the names of video clips (if any) included in this article.
Audio	Similarly, displays a submenu of the names of audios (if any) included in this article.

5.2 Generalizing the Interface

Similar to the system design and functionality, the general features and structure of the user interface can be used for other multimedia information systems. In this case, the menu items and window fields will be modified according to the specific functionality of the target application. In the current implementation, menus are built dynamically to make it easier to modify them according to the application. Examples of such modifications are:

- The **Tools** menu will contain facilities which are related to the target application, instead of the tools provided specifically for news-on-demand. In case of online libraries, this can include subject index, periodicals index, etc.
- The **Filter** window will show the target application's document attributes.
- The **Search** windows will be modified to fit the target application. For example, the **Text** elements menu will contain the text elements contained in the application's DTD (instead of those of news-on-demand).
- Similarly, the **Style Sheet** window and the **Media Settings** window will be modified.

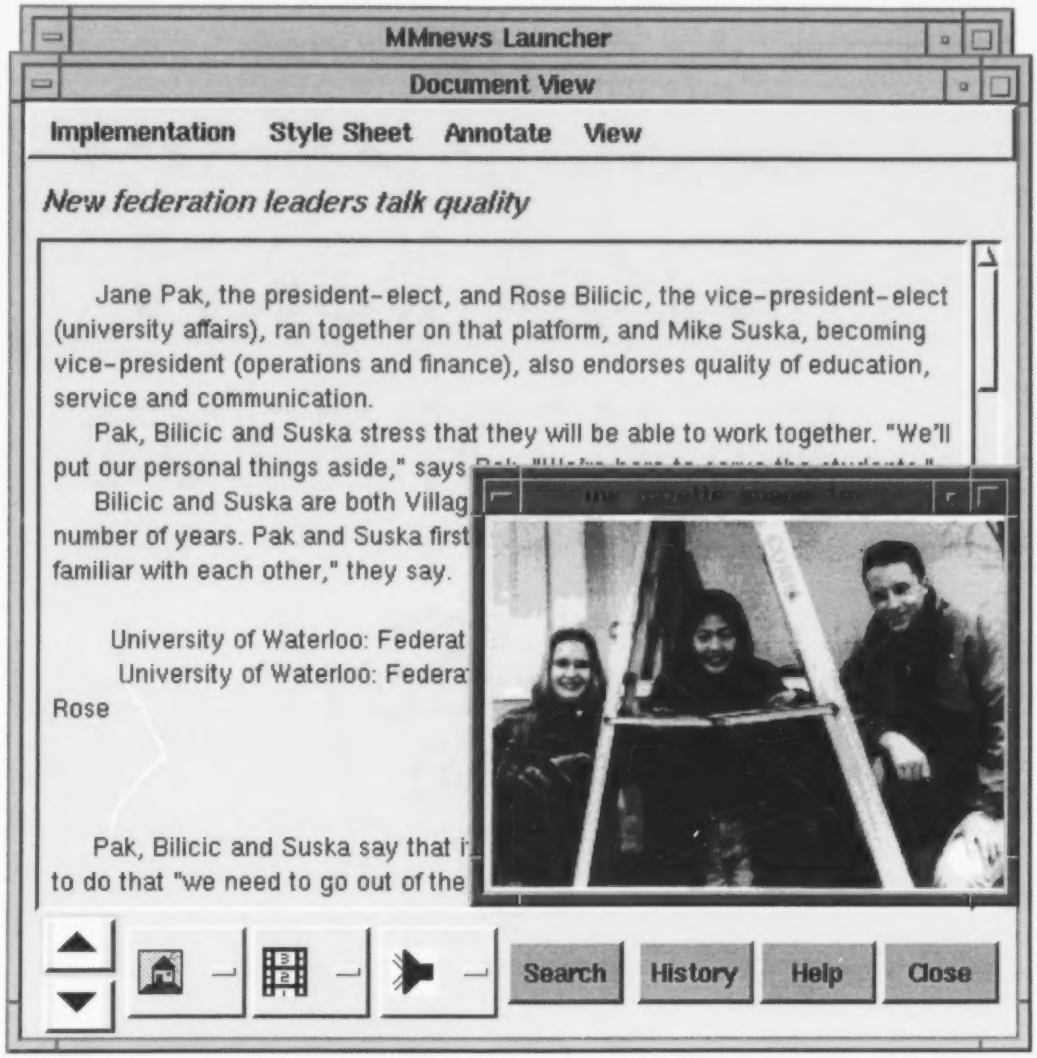
MMnews Launcher
Document View
Implementation
Style Sheet
Annotate
View
New federation leaders talk quality
Jane Pak, the president-elect, and Rose Bilicic, the vice-president-elect (university affairs), ran together on that platform, and Mike Suska, becoming vice-president (operations and finance), also endorses quality of education, service and communication.
Pak, Bilicic and Suska stress that they will be able to work together. "We'll put our personal things aside," says
Bilicic and Suska are both Villag
number of years. Pak and Suska first
familiar with each other," they say.
University of Waterloo: Federat
University of Waterloo: Federa
Rose
Pak, Bilicic and Suska say that i
to do that "we need to go out of the
Search
History
Help
Close

Figure 17: Viewing an Image

Chapter 6

The Multimedia Type System

This chapter summarizes the important features of the design of the multimedia type system for the news-on-demand application. A full description of the type system is available in [Vit95]. Four major issues were considered when designing the multimedia database:

- The different media components of the document (*monomedia objects*) need to be modeled and stored in the database.
- A representation of the document's logical structure is also needed.
- The spatial and temporal relationships between the different monomedia objects need to be represented in the database. This information is needed for the presentation of the multimedia documents.
- The meta-information needed for the operation of the different system components should also be stored in the database.

The logical design of the database uses an object-oriented approach, and follows the SGML/HyTime document standard. SGML [ISO86] deals with textual documents whereas HyTime [ISO92] provides the hypermedia support.

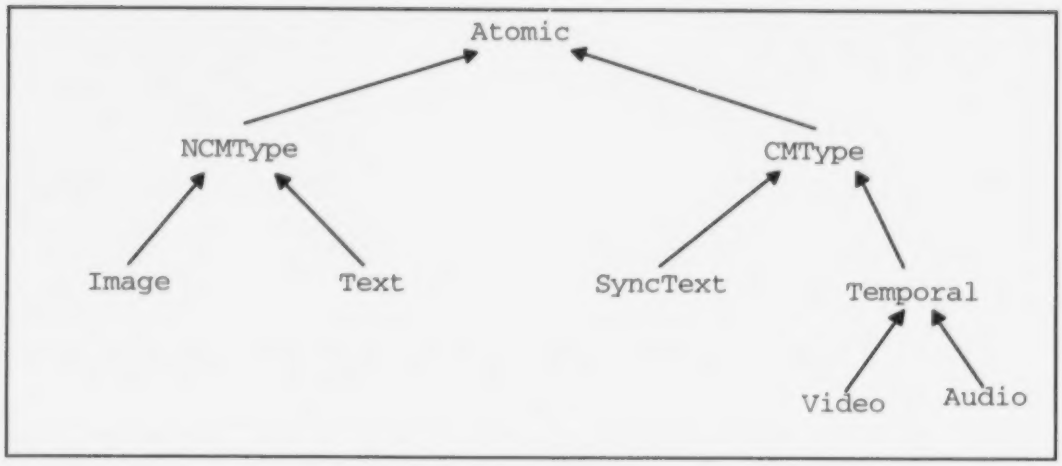

Figure 18: Atomic Types Hierarchy

6.1 Modeling of Monomedia Objects

6.1.1 Atomic Types

Instances of the atomic types hold the raw (mono) media representation together with other parameters. In case of non-continuous media (NCMType), such as text and images, the actual data is stored in the database. In case of continuous media (CMType), such as audio and video, only meta-information is stored in the database whereas the actual data is stored on the continuous media file server. Other parameters related to the quality-of-service (QoS) and synchronization is also stored. The type hierarchy for the atomic types is shown in Figure 18.

6.1.2 Storage Model for Text

In the news documents, the text component of the article is richly structured, consisting of many *elements* such as: paragraphs, sections, emphasis,...etc. Instead of fragmenting the text and storing it with each of these elements (which affects performance and poses several other problems), we store the entire text of a document as a single string. Particular element instances are then associated with their text by storing the start and end positions of their text portion with respect to the document's

entire text string. These pairs of integers (positions) are called *annotations*. Every document instance in the database has a base object which points to the document's text string and the list of annotations on it. This approach allows for a fast and efficient way for searching and displaying text. However, updates to the text content are costly as they may require updating all the annotations. A solution to that problem is to have the annotations relative to some structure (such as a paragraph or section) rather than to the start of the whole text string.

6.2 Modeling of Document Structure

6.2.1 SGML Markup and Elements

SGML is a language which describes the logical structure of a document by using *markups* to mark the document's logical elements. Hierarchies of elements can be formed. A *Document Type Definition* (DTD) must be specified to determine the element types in a document, the relationships between them, and the attributes associated with them. In case of the news-on-demand application, a DTD for multimedia news articles is defined. Examples of logical elements in the defined DTD are: article, headline, author, date, paragraph, figure, figure captions, etc.

Hypermedia support is provided by HyTime which defines 69 special hypermedia elements, called *architectural forms* (AF) that can be used in DTDs. These AFs define links, temporal relationships, events in time, etc.

6.2.2 Type System for Elements

The type system for the logical document elements is shown in Figure 19, 20, and 21. Element is subtyped into TextElement, Structured, and HyElement. The TextElement is the supertype for all the textual elements in the DTD that have no subelements (simple text elements). The Structured is the supertype for textual elements with

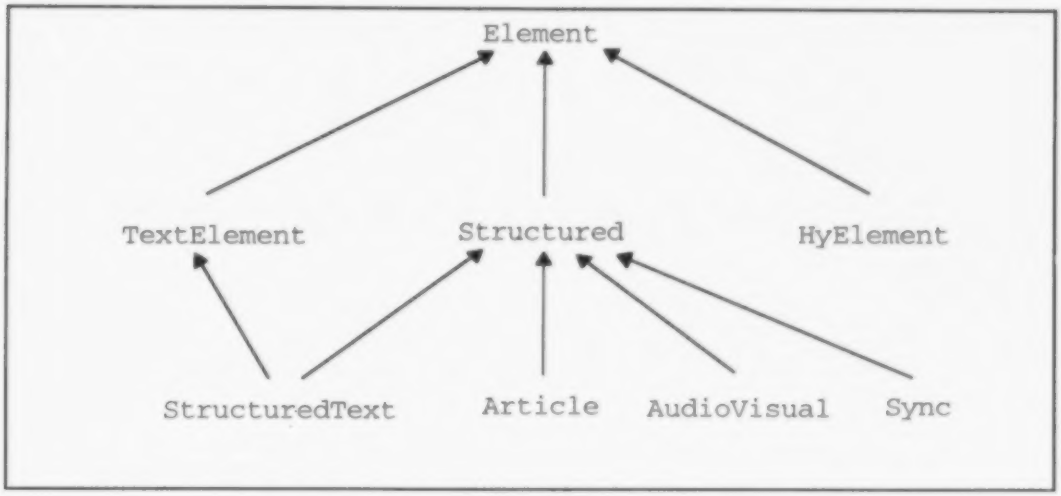

Figure 19: First-Level Element Type Hierarchy

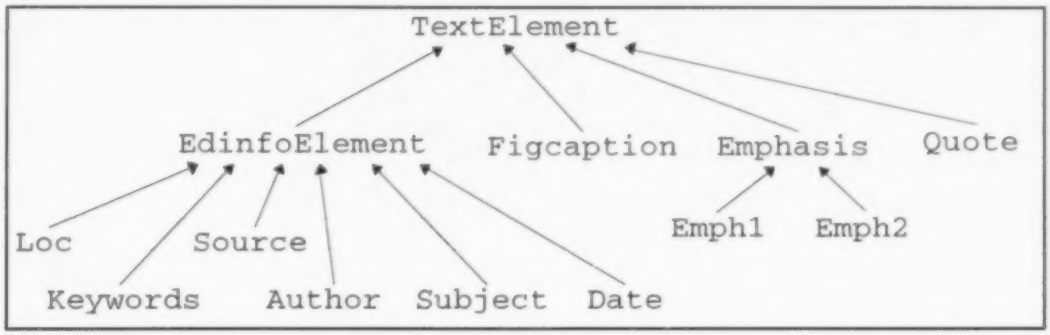

Figure 20: Type Hierarchy for Other Text Elements

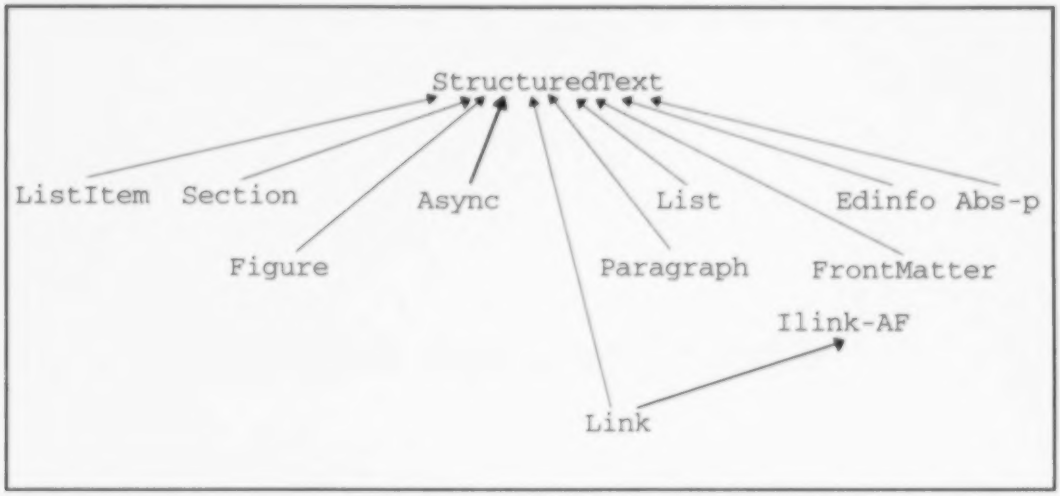

Figure 21: Type Hierarchy for Structured Text Elements

complex content models. These usually have subelements. The HyElement abstracts all the HyTime elements. All elements maintain a link to their parent element, as well as a link to the article that contains them.

6.3 Modeling of Presentation Information

As has been mentioned earlier, the spatial and temporal relationships between the document elements should be represented in the database. This information is used by the synchronization routines to retrieve and present these related objects according to a presentation scenario. The HyTime standard is followed when defining this representation.

To represent relatively simple spatial and temporal constraints between document elements, the *finite coordinate space* (FCS) architectural form is used. A finite coordinate space is a set of axes modeling space and time. The FCS we use has three axes: two for space and the third for time. An extent on the FCS is a set of ranges along the various axes defined. An *event* is modeled as an extent on the FCS. The document instance associates a data object with an event. The semantics and the

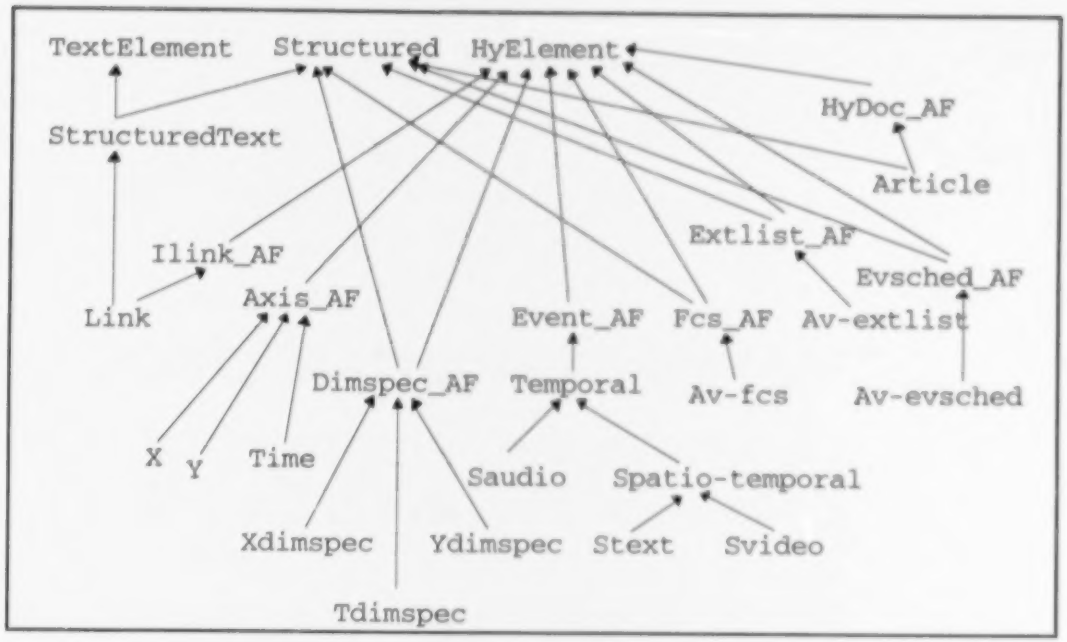

Figure 22: Type Hierarchy for HyTime Elements

manner in which the events are rendered are defined by the application.

The HyTime elements in the type system are shown in Figure 22. Currently, the visual querying interface does not heavily interact with the HyTime elements. For that reason, these elements are not discussed here. For a detailed description of them, please refer to [Vit95].

Chapter 7

Querying - Linking to the Database

A unique feature of the visual query facility is its tight integration with the multimedia database system. Each user action results in one or more queries to be issued to the underlying database which is managed by ObjectStore [OHMS92]. The visual query facility, running on client machines, issues ObjectStore queries to fetch the required information from the server. The ObjectStore client then returns the matching objects to the visual query facility. It is then the responsibility of the visual query interface to manipulate these objects depending on the task to be achieved.

This chapter first discusses ObjectStore queries in general, and then, specifically, explains the categories of queries used in the news-on-demand application. Example queries are given to clarify the discussion.

7.1 ObjectStore Queries

ObjectStore provides query processing facilities to make possible the associative access of data objects which is needed by many applications [Obj94a, Obj94b]. The ObjectStore *query optimizer* produces efficient search strategies to minimize the number of objects examined in response to a query.

A query is performed on a collection of data objects of type **os_Collection** or any of its subtypes. Calls to the query function look as follows [Obj94b]:

```
collection-expression.query (
   element-type-name,
   query-string,
   schema-database)
```

where the **collection-expression** is the collection over which the query will execute. The **element-type-name** is a string indicating the element type of the collection being queried. The **query-string** is an expression indicating the query selection criterion. An element satisfies the selection criterion if the control expression evaluates to a nonzero int; otherwise the element does not satisfy the criterion. Query strings can contain any integer-valued C++ expression as long as there are no variables which are not data members, and there are no function calls except to **strcmp**. The **schema-database** is the database in which the queried collection resides.

An example query that returns a collection of teenagers among the elements of the set people can be performed as follows [Obj94b]:

```
os_database *people_database;
os_Set<person*> *people;
...
os_Set<person*> *teenagers =
   people->query (
      "person*",
      "this->age >= 13 && this->age <= 19",
      people_database);
```

The above **query()** function returns a collection that is allocated on the heap. In case no elements satisfy the selection criteria, an empty collection is returned.

Two other types of ObjectStore queries are worth mentioning here [Obj94b]:

- Single Element Queries

  Some queries are intended to return one element, rather than a collection of elements. The os_Collection::query_pick() is used, in that case, to return a single element. This allows for more opportunities for query optimization. If more than one element satisfies the query, one element will be chosen at random and returned. Calls to query_pick() are similar to calls to query().

- Existential Queries

  Existential queries are used to determine whether there exists some element that satisfies the query selection criterion in the queried collection in cases when there is no interest in the identity of such an element (or elements). The os_Collection::exists() provides this functionality. Calls to exists() are similar to calls to query() and query_pick(). However, it returns a nonzero int in case of true and 0 in case of false.

The above query functions are used when the selection criterion is fixed and all the values are known. In cases where this information changes or when string comparisons are part of the selection criterion, pre-analyzed queries must be used.

Pre-analyzed Queries

When a query is performed several times, perhaps with different values or on different collections, a *pre-analyzed query* should be used to reduce the query analysis cost by only analyzing the query once instead of several times. To do this, an object of type os_coll_query is created. The query is analyzed once upon creation. Every time the query is performed, bindings for the free variables and the function calls, as well as the collection over which the query will run, should be provided. To create a pre-analyzed query, one of the following member functions is used: os_coll_query::create(), os_coll_query::create_pick() , and os_coll_query::create_exists(). These are equivalent to the query types discussed earlier. The create functions look as follows [Obj94a]:

```
const os_coll_query &create (
```

```
element-type-name,
query-string,
schema-database,
cache_query=0)
```

The **element-type-name**, **query-string**, and **schema-database** follow the same rules as parameters explained earlier. The **query-string**, in case of pre-analyzed queries, can also include calls to non-overloaded global functions, provided that:

- the return type of the function is explicitly specified by a cast,
- the function references are bound during query binding time, and
- all function calls involve zero, one, or two arguments, and in case of two arguments, the first one is a pointer.

Variables can also be included in the **query-string** as long as the type of each variable (except data members) is explicitly defined by a cast, and bound during query binding.

The **cache-query** is an optional flag indicating whether the query object is created persistently in the **schema-database** or transiently allocated. If **cache-query** has zero int value, the query object is created transiently (the default); otherwise, it is created persistently.

An example for creating a pre-analyzed query for people in a given age range is as follows [Obj94b]:

```
const os_coll_query &age_range_query =
   os_coll_query::create (
      "person*",
      "age >= (int) min_age && age <= (int) max_age",
      people_database);
```

Subsequently, anytime one needs to run this query, a *bound query* (**os_bound_query**) is constructed to provide the necessary bindings for the pre-analyzed query:

```
int teenage_min_age = 13, teenage_max_age = 19;
os_bound_query teenage_range_query (
   age_range_query,
   (
       os_keyword_arg("min_age", teenage_min_age),
       os_keyword_arg("max_age", teenage_max_age)
   )
);
```

This creates a bound query for finding teenagers using the previously analyzed query `age_range_query`. The bound query is then used to evaluate the query in the usual manner:

```
people.query (teenage_range_query);
```

In case of function calls in the **query-string**, bindings are also provided for the function call as well as its arguments (if any) in the bound query.

7.2 The Type System and Queries

The multimedia type system, described in the previous chapter, is implemented using C++ and ObjectStore. Each type corresponds to a C++ class. Instances of types/classes are made persistent using ObjectStore.

The extents of objects of each type are maintained to allow queries to search objects of a particular type. Type extents are implemented as persistent parameterized collections with the type as a parameter. Persistent names are given to the extents to identify them as *database roots*. The name for a type extent is given as the type name, followed by the string `"_extent_root"`. For example, the **Article** extent is named `"Article_extent_root"`. When an object of a particular type is instantiated, a reference to that object is also inserted in its type's extent.

In order to perform a query on objects of a particular type, the extent of this

type must be fetched from the database. For example, to fetch the Article extent, the application does the following:

```
os_typespec *Article_extent_type
   = os_Set<Article*>::get_os_typespec();

os_Set<Article *> *Article_extent
   = (os_Set<Article*>*) (db->find_root("Article_extent_root")
     ->get_value(Article_extent_type);
```

Subsequently, a query can be performed on the Article_extent, which is the collection of all articles in the database, to get a subset of that collection.

Due to the nature of the news-on-demand application which requires that queries be performed several times with different string values, pre-analyzed queries are created, and bindings are provided before the query is executed on some collection.

For example, to perform a query to fetch all the articles whose location (which is an attribute of Article) is "Edmonton", a pre-analyzed query to fetch all the articles whose location matches a specific string is created persistently and a reference to the query object is kept:

```
articleLocQuery = & os_coll_query::create (
   "Article*",
   "!strcmp (location, (char *) string)",
   news_database, 1);
```

Subsequently, to fetch all the articles whose location is the string "Edmonton", the following bound query is provided and used to perform the query:

```
strcpy (loc_string, "Edmonton");

os_bound_query location_query (
   *articleLocQuery, (
```

```
        os_keyword_arg ("string", loc_string)));

    os_Set<Article*> *matching_articles
        = Article_extent->query (location_query);
```

For any subsequent queries of the same types (i.e., on article location), a different binding needs to be provided before executing the query.

An alternative to storing the query object persistently and subsequently referencing it is to create it transiently every time you need to bind and execute a query. Either of these implementation choices can be followed. Tests to determine performance implications showed that there is hardly any difference in performance between the two methods.

Extents for all the types in the type system are maintained and queried in the same fashion as will be explained in the following section.

7.3 Classes of Queries

As has been mentioned earlier, all user interactions with the system are translated into ObjectStore queries that are executed on the database to fetch the requested results. These queries can be classified in three main categories:

7.3.1 Queries on Articles' Attributes

Queries on articles' attributes return collections of articles whose attributes match the specified values. An example of this category of queries is querying articles whose location string is "Edmonton" which was discussed earlier.

Other article attributes include title, category, source, authors, keywords, and date. For each attribute, a pre-analyzed query (similar to `articleLocQuery`) is created. Bindings are provided before executing the query.

These queries are used in filtering operations. Filters contain values which match some or all the different attributes of articles. A filtering operation involves going over the different values in a filter and using these values to provide bindings for the corresponding queries. The result is a collection of articles whose attributes match the values specified in the filter.

7.3.2 Queries on User Profiles

As has been mentioned earlier, in addition to news articles, the multimedia database stores the user profiles for users of the system. It is useful to refer to Appendix A for a detailed description of the user profile attributes and member functions.

The extent of user profiles is maintained and fetched in exactly the same fashion as the Article's extent explained earlier. Every user profile is uniquely identified by its username. One type of query is performed on the extent of user profiles to return the user profile whose username matches a given user. And since it is known that there can be at most one user profile that matches the search criteria, the query_pick() is used to allow for more optimization. The pre-analyzed query is defined as follows:

```
userProfileNamePick = & os_coll_query::create_pick (
    "UserProfile*",
    "!strcmp (userName, (char *) string)",
    news-database, 1);
```

When starting a multimedia news session, the user enters his/her username and the news database file he/she wishes to connect to (Chapter 5). The system opens the specified database file and proceeds to fetch the user's profile. This is done by binding the userProfileNamePick query using a bound query to the given username as follows:

```
os_bound_query profile_pick (
    userProfileNamePick, (
    os_keyword_arg("string", currentUserName)));
```

```
currentUserProfile = Profile_extent->query_pick (profile_pick);
```

If the querying operation returns NULL, it means that the profile with the specified username does not exist, the system creates a new user profile for this new user. After fetching the user profile, a reference to it is kept throughout the session.

Subsequently during a session, two other types of queries can be executed on the `userStyleSheets` collection and the `userFilters` collection which are data members in the user profile. Each style sheet and/or filter is identified by a unique name which is used when querying the collections. The query scenario is similar to querying the user profile extent discussed above.

7.3.3 Search Queries

Search queries are queries focusing on content rather than attributes. The system allows users to search documents as well as document components such as: abstracts, paragraphs, sections, quotes, images, audio, video, editorial information, etc. Users also specify the scope of the search which can be all the documents in the database, the documents in the current list, or the current document. Also, the types of objects returned by the search is specified by the users to be either complete documents, document components or both.

Searching Complete Documents

As has been mentioned in the previous chapter, the text string of any article is stored as one string associated with each Article instance in the database. If users specify that they wish to search whole documents for a specific string and get back complete documents (and not components), the system performs pattern matching on the text string of the article instead of searching all the individual components to reduce unnecessary overhead, and returns the articles that contained the given string.

This is done by iterating over the `ArticlesToSearch` collection using an ObjectStore os_Cursor. Each article in the collection is checked and the matching elements are inserted in the returned collection of articles. No other query functions are used in this case.

Searching Text Elements

Text Elements (depicted in Chapter 6 Figures 20 and 21) include abstracts, paragraphs, sections, headlines, quotes, emphasis elements, and editorial information.

Headlines and editorial information such as authors, keywords, etc. are modeled as attributes (data members) of Article. Therefore, they are searched as has been explained earlier in Section 7.3.1.

Similar scenarios are followed when searching the other text elements. Therefore, an example of searching one text element will be given here. The others follow the same way.

Assume the user wants to search paragraphs. Two pre-analyzed queries are needed to make it possible to search for a particular string within instances of type Paragraph. The first query is executed over the Paragraph_extent in the database and returns the paragraphs whose text strings include the string the user is searching for:

```
paragraphSearchQuery = & os_coll_query::create (
    "Paragraph*",
    "(int) includes ((char *)getStr(this), (char *)string",
    news_database, 1);
```

The above query returns all the matching paragraphs in the database. However, the user is only interested in the paragraphs which belong to the documents defined in the search scope. For that reason, the collection returned by the first query is used by the second query which returns a subset of this collection containing the paragraphs that belong to the articles of interest (according to the specified search

scope). However, in case the search scope is all the documents in the database, the second query is bypassed.

```
paragraphInArticleQuery = & os_coll_query::create (
    "Paragraph*",
    "articleElement == (Article *) anArticle",
    news_database, 1);
```

After looking at the query objects, it is useful to discuss the bindings which are needed to perform the search. The `paragraphSearchQuery` is bound as follows:

```
os_bound_query paragraph_query (
    *paragraphSearchQuery, (
    os_keyword_arg ("string", stringToSearchFor),
    os_keyword_arg ("getStr", getParagraphStr),
    os_keyword_arg ("includes", includes)));


os_Set<Paragraph*> matching_paragraphs
    = Paragraph_extent->query (paragraph_query);
```

There are two function bindings and one variable binding. The `stringToSearchFor` is the string the user specified during the search (optionally including Boolean combinations). `getParagraphStr` is a global function that returns the text of its paragraph instance argument (using annotations). Finally, `includes` is a global function which takes two arguments the text of the paragraph and the `stringToSearchFor` and performs pattern matching between its two arguments, taking into account the Boolean combinations of strings which might be included in the `stringToSearchFor`. It returns nonzero if there is a match, otherwise a 0 is returned. After providing the bindings, the bound query is executed on the `Paragraph_extent`.

In case the search scope is not all the documents in the database (in which case the `matching_paragraphs` collection is returned by the search), the second query is also bound and executed as follows:

```
os_Cursor<Article*> aCursor(articlesToSearch);
Article *article;

for (article=aCursor.first(); article; article=aCursor.next()) {
   os_bound_query paragraph_inArticle (
      *paragraphInArticleQuery, (
      os_keyword_arg ("anArticle", article)));

   selected_paragraphs |=
      matching_paragraphs.query (paragraph_inArticle);
};
```

The application iterates over the set of `articlesToSearch` which is determined by the search scope. Each article instance in the collection is used as a binding for the `paragraphInArticleQuery` to return a collection the paragraph instances which belong to this article. The returned collections are unioned to form the `selected_paragraphs`. Thus, the matching paragraphs which belong to articles outside the search scope are discarded. This has to be done this way because ObjectStore does not provide a join facility which would allow for the joining of two collections.

Searching Other Media

Currently, a keyword search is provided to search images, audio, and video objects. A textual description is associated with audio and video instances. In case of images, the figure caption is used as the description of the image. Textual descriptions and/or figure captions are searched the same way text elements are searched, and the matching objects are returned.

Ultimately, we are interested in providing more powerful search capabilities for these media types, including content-based searching and indexing of images.

7.4 Implications of Using ObjectStore

The implementation of the queries is limited by the capabilities provided by ObjectStore as well as the design of the type system (which is in turn also affected by ObjectStore).

Queries are performed on collections of a particular type. For that reason, extents of all the different types in the type system (implemented as ObjectStore parameterized collections) are maintained to allow for querying on a particular type. A persistent object/collection in ObjectStore can be accessed by navigation from another object or through database entry points *(database roots)*. Therefore, type extents are defined as database entry points to allow them to be easily fetched for subsequent queries.

ObjectStore does not support any pattern matching capabilities except for `strcmp`. The query implementor has to implement routines that employ any other search techniques to be used as a selection criterion in queries. Any such routine must be a free (global) function with at most two parameters and returing an int-value. Bindings for each function need to be provided before executing the query. Therefore, all the search routines were implemented following these constraints.

Another disadvantage is the need to perform a lot of client processing of persistent collections or objects when performing certain tasks. For example, in case of searching text elements or continuous media descriptions, first the extents of the types to be searched are queried to get the matching objects. The returned collection contains the matching objects of a certain type in the whole database. Further client processing must be performed to discard objects which belong to articles outside the search scope since ObjectSore does not provide the facility to join two collections. This processing involves iterating over the collection of articles in the search scope and performing a query to determine the collection of objects which belong to that article. The resulting collections are unioned together to get a collection of all objects in the search scope (See example of searching paragraphs discussed in the previous section). This task involves a lot of client processing which cannot be optimized by

the DBMS on the server side.

Furthermore, ObjectStore poses other problems for supporting multimedia applications as it does not provide any inherent multimedia support such as support for modeling spatial and temporal relationships or real time constraints on the delivery of continuous data for synchronization. The current design of this project does not integrate the continuous media file server with the database (Chapter 3). The database only stores meta-information in the case of continuous media, and thus does not deal with synchronization and real-time delivery issues. Therefore, these issues were not investigated in the framework of ObjectStore. However, it is expected that ObjectStore's functionality cannot (yet) provide adequate solutions for these problems.

Chapter 8

Implementation Issues

The visual query facility is made up of two components: a Smalltalk user interface and a C++ query agent both running on the same client machine (Figure 23). A pipe is set up between these two components and they communicate via a string interface. The user interface is responsible for handling all interactions with the user. User actions are translated by the Smalltalk interface into string commands which are understood by the C++ query agent. It then communicates with the C++ program which generates queries on the ObjectStore database. After executing the queries and getting back the results from the server, the C++ program returns the results to the Smalltalk interface. It is again the interface's responsibility to display these results to the user.

8.1 The Smalltalk User Interface

The Smalltalk user interface is implemented using ParcPlace Smalltalk/VisualWorks release 2.0 [GR85, Par94a, Par94b] for the IBM RS/6000 machines.

The interface can be divided into two main subsystems (Figure 24):

- **User Interface Sub-system**

  The user interface subsystem is responsible for all the user interactions. It con-

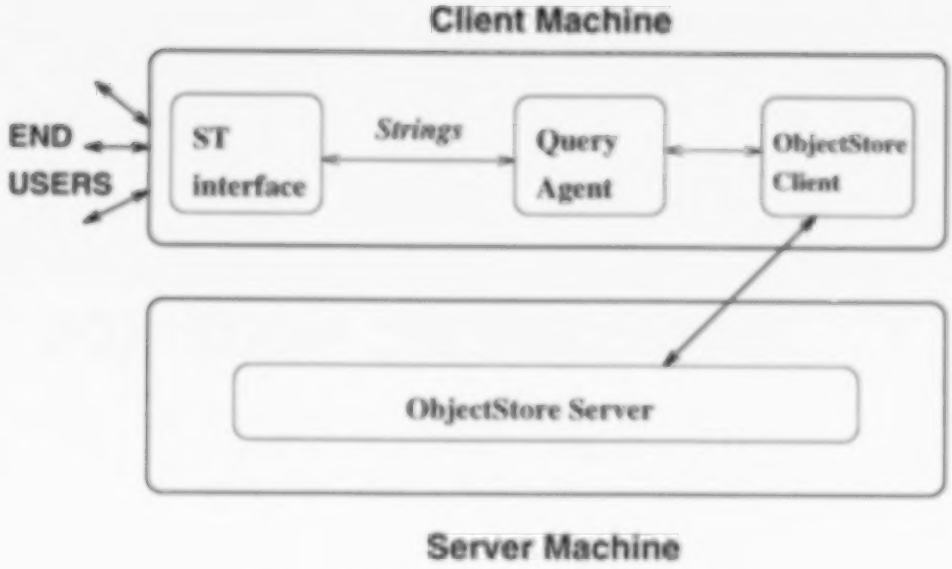

Figure 23: Visual Query Components

tains all the application model classes which are responsible for the interface windows, as well as the data model classes which are responsible for modeling different entities in the system such as documents, filters, style sheets, etc. Usually every instance of an application model class is associated with an instance of a data model class which contains necessary information needed for presentation and interaction with the user. For example, a MMFilter object is associated with each MUIFilter object which handles the filter view.

- **Communication Subsystem**

  The communication subsystem is responsible for sending commands and receiving results from the C++ program. It consists of four main classes:

  - **Session:** is the class with which the user interface subsystem interacts to send and receive information from the C++ program. Every news-on-demand session uses one instance of that class which contains the global settings of that session. These include the username, database filename, the global media settings, and the connection which maintains the pipe.

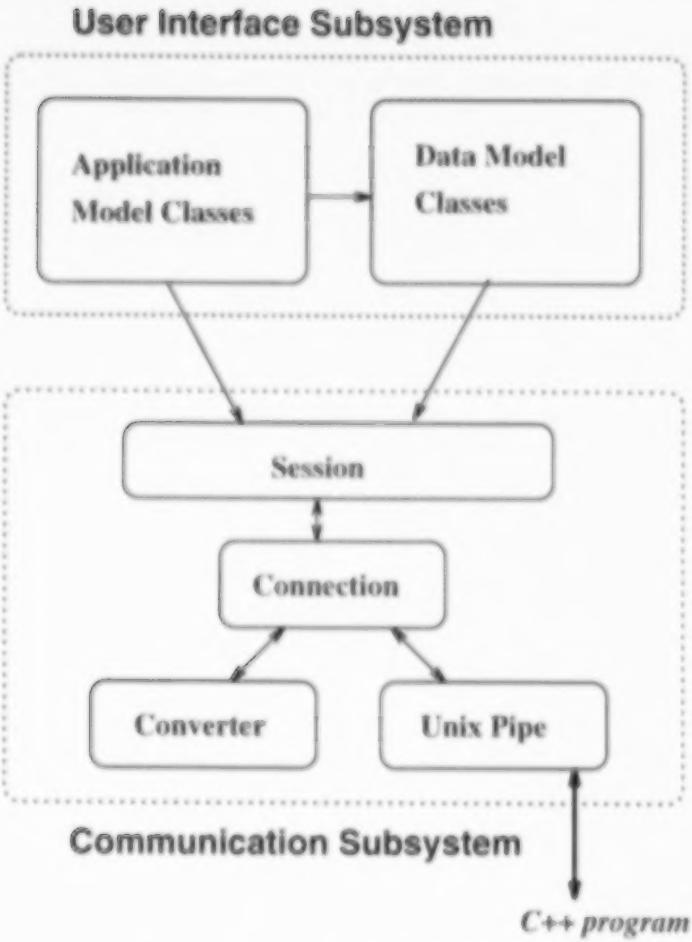

Figure 21: Smalltalk Main Modules

- **Connection:** handles the interaction between the session and the underlying communication classes, namely the Converter and the Unix Pipe.
- **Converter:** converts the Smalltalk commands and objects into the corresponding strings which are understood by the C++ program. For a list of these strings, please refer to Appendix B.
- **Unix Pipe:** maintains the pipe to the C++ program. During a news session startup, the pipe is given the name of the C++ executable program. It starts it as a process and maintains a pipe to it. The C++ program only exits when a quit command is sent from the Smalltalk side through the pipe.

8.2 The C++ Query Agent

The C++ query agent is implemented using the xlC product from IBM and ObjectStore on the IBM RS/6000 machines.

It consists of the following main classes (Figure 25):

- **Pipe:** is responsible for handling the input and output to the pipe connected to Smalltalk. Currently, standard input and output are used for this communication. However, this class can be implemented using sockets or any other communication protocol.
- **Parser:** parses the input data (commands) sent by the Smalltalk interface and generates an action code accordingly. A list of valid commands are given in Appendix B.
- **DBmanager:** is the main class responsible for interacting with the ObjectStore database and returning the results via the pipe.
- **Search:** is a supporting class responsible for performing searches due to their complex nature.

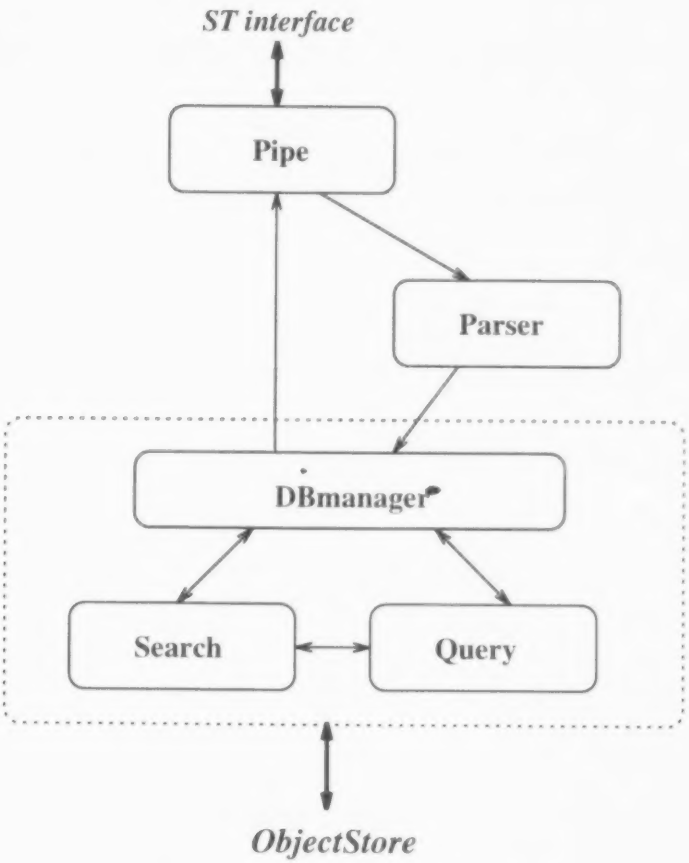

Figure 25: C++ Main Modules

- **Query:** maintains all the persistent query objects which were discussed in the previous chapters.

- **Others:** Other classes are responsible for maintaining and handling specific tasks related to the database and are used by the **DBmanager** class.

When the Smalltalk interface starts up the query agent as a process on the same machine, the main program listens to input from the **pipe**. Once a command delimiter is reached, this signals the end of a command. The **parser** then parses the command and generates an action code. According to this action code, the **DBmanager** executes queries on the ObjectStore database, and then returns the results to Smalltalk via the **pipe**. In case of invalid commands, an error string is sent to Smalltalk. After sending back the results, the program continues to listen for input.

8.3 Alternatives for Implementation

The reason for implementing the visual query facility this way is that the ObjectStore database cannot be accessed directly from Smalltalk. It is necessary to access the ObjectStore database from a C++ program. Attempts to link Smalltalk directly to the C++ program using the C Connect interface (available with ParcPlace Smalltalk) also failed.

Therefore, the two components communicated via a string interface. An object-oriented interface could have been provided. However, the entire type system would have been duplicated on the Smalltalk side. Since there is no automatic way to do that, any change in the type system on the C++ side would result in a modification to the Smalltalk implementation which could result in maintenance problems. Furthermore, this approach defeats the ObjectStore philosophy which fetches objects and components from the server to the client only when needed. For these reasons, a string interface which hides specific implementation details of the type system from

the Smalltalk side and only sends needed information for presentation and user interaction purposes was chosen. A major advantage to this approach is that the C++ program can be used with any interface that communicates following the same format. Details of the underlying database design are completely transparent to the interface. Likewise, if the underlying database and the C++ program change, there is no need to change the user interface as long as the string interface remains unchanged. A disadvantage is the overhead encountered from parsing input on both sides and fetching additional information in some cases. Some work can be done to optimize the communication between the two components.

A completely different alternative to this approach is to develop the user interface in C++ using the AIX Interface Composer. This alternative was not chosen due to time constraints and the high learning curve of the AIX interface builder as compared to the relatively easier Smalltalk interface builder.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

This thesis describes a visual query facility, built on top of a distributed multimedia database, to allow users to access the database. This facility concentrates on providing *querying* capabilities which are necessary to allow users to directly and efficiently retrieve needed information from the database. Some *browsing* capabilities are also provided to allow users to browse through information in the database. The target application is news- on-demand, a distributed multimedia news application. However, the same design and implementation principles of the visual query facility can be easily extended and applied to other multimedia applications such as online museums and multimedia library systems. Thus, the issues investigated in this thesis can contribute to user interfaces of a variety of multimedia information systems.

The identifying and novel features of this work are:

- The tight integration of the user interface with a multimedia database.
- A rich querying facility coupled with a browsing facility.
- A visual interface that relieves users from having to type ObjectStore queries (which can be difficult).

- The separation of the logical document content from presentation (by making use of style sheets) to provide a completely customizable system to the user.

ParcPlace Smalltalk/VisualWorks and ObjectStore were used for the implementation of the system. ParcPlace Smalltalk/VisualWorks provided an effective tool for building the graphical user interface. A prototype of the system was built early on in the project. The system then evolved gradually from the initial prototype as requirements and specifications became clearer.

9.2 Unimplemented Features

Some of the features included in the design of the user interface are not implemented in this version. These features are:

- A hypermedia browser which enables users to navigate from one news document to others.
- User Annotations which allow users to add their own comments on the news items they read and relate them to other items.
- Hotlist which allows users to add articles of interest to their hotlist for faster access.
- Online Documentation: Although context-sensitive help for all MMnews windows is provided, an online hypertext help document is not implemented in this version.

9.3 Future Enhancements

This work is only an initial step in investigating the development of query languages, access primitives, and visual query facilities that allow for sophisticated querying of multimedia databases. Thus, the visual query facility can be enhanced in a variety of ways:

- A first step towards enhancement is to implement the features discussed in the previous section to provide a richer more complete query interface for news-on-demand users.

- An interesting enhancement would be to provide the querying facility through commonly used browsers such as Netscape. Due to the division of the query facility into the Smalltalk user interface and the C++ query agent, it would be easy to substitute the Smalltalk user interface with Netscape forms that can then communicate with the query agent using the same string interface. There would be no need to change anything on the query agent side. Other user interfaces can also be used as long as they communicate with the query agent using the defined string communication interface (Appendix B).

- Currently, the visual query facility allows users to access information stored in one ObjectStore database file. Providing access to multiple databases would provide added value to the system. Adding this capability is very straightforward. The user interface would allow users to specify multiple databases which would be opened and maintained by the query agent. Each string request sent from the user interface will have to specify the target database(s) to perform the query on.

- As it is now, the visual query facility provides complicated content searches for text elements in a document. In case of other media types, only limited keyword searches are provided. An interesting enhancement would be to provide more sophisticated searches for different media types. This includes providing, in the long run, content-based indexing and querying of images. Extensions to video searching can include providing annotations for video scenes instead of one textual description of the whole video clip. Searching audio can also be enhanced by providing textual scripts of the audio recordings (possibly generated automatically through speech recognition technology).

- The visual query facility is tightly integrated with a specific multimedia type system [Vit95], generated from a specific DTD for news articles. As we go to the meta level to provide corresponding type systems for different DTDs and applications, parallel work can be done on the visual query facility to produce an equivalent querying interface for the different type systems supported. Providing a querying interface that spans over multiple DTDs is another interesting step in this direction. However, it poses several questions in terms of the generality needed to achieve such a task.

- Furthermore, the querying facilities provided are limited by the support provided by ObjectStore which is a closed DBMS with no inherent support for multimedia. Research work is currently conducted at the Laboratory for Database Systems Research, University of Alberta, to develop an extensible OBMS that has inherent multimedia support. In the long run, TIGUKAT will replace ObjectStore. With the development of this open system, investigating more sophisticated query facilities and languages will be possible.

Bibliography

[ACMZ91] E. Andonoff, M. Canillac, C. Mendiboure, and G. Zurfluh. Hypertext interface for an object-oriented database. In *Intelligent Text and Image Handling. RIAO '91*, pages 843–862, 1991.

[ACMZ92] E. Andonoff, M. Canillac, C. Mendiboure, and G. Zurfluh. OHQL: A hypertext approach for manipulating object-oriented databases. *Information Processing & Management*, 28(5):567–579, 1992.

[Ada93] J.A. Adam. Interactive multimedia: Special report. *IEEE Spectrum*, pages 22–39, March 1993.

[ADE93] I. B. Arpinar, A. Dogac, and C. Evrendilek. MoodView: an advanced graphical user interface for OODBMSs. *SIGMOD Record*, 22(4):11–18, Dec. 1993.

[AMY88] R. M. Akscyn, D. L. McCracken, and E. A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7):820–835, 1988.

[BD92] M.M. Blattner and R.B. Dannenberg, editors. *Multimedia Interface Design*. New York, NY: ACM Press, 1992.

[Bud91] T. Budd. *An Introduction to Object-Oriented Programming*. Addison Welsey Publishing Co., 1991.

[Cru92] I.F. Cruz. DOODLE: A visual langauge for object-oriented databases. In *Proceedings of the 1992 ACM SIGMOD. International Conference on Management Data.*, pages 71–80, 1992.

[D+91] O. Deux et al. The O2 system. *Communications of the ACM*, 34(10):34–48, October 1991.

[DG92] N. Dimitrova and F. Golshani. EVA: a query language for multimedia information systems. In *Multimedia Information Systems - An International Workshop*, pages 1–20, February 1992.

[Eme89] S. L. Emerson. *The Practical SQL Handbook: Using Structured Query Language*. Addison-Wesley Publishing Company, Inc., 1989.

[Fer94] F. M. Ferrara. The KIM query system. an iconic interface to the unified access to distributed multimedia databases. *SIGCHI Bulletin*, 26(3):30–39, July 1994.

[FS91] H.P. Frei and P. Schauble. Designing a hypermedia information system. In *DEXA 91. Database and Expert Systems Applications*, pages 449–454, 1991.

[GOC+92] C. Goble, M. O'Docherty, P. Crowther, M. Ireton, J. Oakley, and C. Xydeas. The manchester multimedia information system. In *Advances in Database Technology - EDBT '92*, pages 39–55, 1992.

[GR85] A. Goldberg and D. Robson. *SmallTalk-80: The Language and its Implementation*. Addison Wesley, 1985.

[GT94] K. Grønbæk and R.H. Trigg. Design issues for a dexter-based hypermedia system. *Communications of the ACM*, 37(2):40–49, February 1994.

[Hal88] F. G. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, 1988.

[Hay93] R. Haykin. *Demystifying Multimedia.* Apple Computer, Inc., 1993.

[Her94] E. Van Herwijnen. *Practical SGML - Second Edition.* Kluwer Academic Publishers, 1994.

[HK92] K. Hirata and T. Kato. Query by visual example - content based image retrieval. In *Advances in Database Technology - EDBT '92. Proceedings 3rd International Conference on Extending Database Technology*, pages 56–71, 1992.

[HKR+92] B.J. Haan, P. Kahn, V.A. Riley, J.H. Coombs, and N.K. Meyrowitz. IRIS hypermedia services. *Communications of the ACM*, 35(1):36–51, January 1992.

[ISO86] International Standards Organization. Information Processing – Text and Office Information Systems – Standard Generalized Markup Language (ISO 8879), 1986.

[ISO92] International Standards Organization. Hypermedia/Time-based Structuring Language: HyTime (ISO 10744), 1992.

[K+90] W. Kim et al. Architecture of the Orion next-generation database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109–124, March 1990.

[KFS90] T. Kato, K. Fujimura, and H. Shimogaki. Trademark: multimedia image database system with intelligent human interface. *Systems and Computers in Japan*, 21(11):33–46, 1990.

[KKL91] D. A. Keim, K.-C. Kim, and V. Lum. A friendly and intelligent approach to data retrieval in a multimedia dbms. In *DEXA 91. Database and Expert Systems Applications*, pages 102–111, 1991.

[LG94] L. Li and N. Georganas. MPEG-2 coded- and uncoded-stream synchronization control for real-time multimedia transmission and presentation

over B-ISDN. In *ACM Multimedia 94. Proceedings of Second ACM International Conference on Multimedia*, pages 239–246, 1994.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore database system. *Communications of the ACM*, 34(10):50–63, October 1991.

[Luc90] D. Lucarella. A model for hypertext based information retrieval. In *Hypertext: Concepts, Systems and Applications. Proceedings of the European Conference on Hypertext*, pages 81–94, November 1990.

[MS87] D. Maier and J. Stein. *Research Directions in Object-Oriented Programming*, pages 355–392. MIT Press, Cambridge, MA, 1987.

[Nie90a] J. Nielsen. *Hypertext & Hypermedia*. San Diego, CA: Academic Press, Inc., 1990.

[Nie90b] J. Nielsen. The art of navigating through hypertext. *Communications of the ACM*, 33(3):296–310, March 1990.

[Nie91] J. Nielsen. Usability considerations in introducing hypertext. In H. Brown, editor, *Hypermedia/Hypertext And Object-Oriented Databases*, pages 3–17. Chapman & Hall, 1991.

[Obj94a] Object Design, Inc., Burlington, MA, USA. *ObjectStore Reference Manual for OS/2 and AIX/xlC Systems*, January 1994.

[Obj94b] Object Design, Inc., Burlington, MA, USA. *ObjectStore User Guide for OS/2 and AIX/xlC Systems*, January 1994.

[O'D93] M. H. O'Docherty. A multimedia information system with automatic content retrieval. Master's thesis, Victoria University of Manchester, Department of Computer Science, February 1993.

[OHMS92] J. Orenstein, S. Haradhvala, B. Margulies, and D. Sakahara. Query processing in the ObjectStore database system. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data.*, pages 403–412, 1992.

[Ont91] Ontologic Inc., Burlington, MA, USA. *ONTOS Developer's Guide, Version 2.0*, February 1991.

[Par94a] ParcPlace Systems, Sunnyvale, CA, USA. *VisualWorks User's Guide*, August 1994.

[Par94b] ParcPlace Systems, Sunnyvale, CA, USA. *VisualWorks Cookbook*, July 1994.

[Pla91] C. Plaisant. An overview of Hyperties, its user interface and data model. In H. Brown, editor, *Hypermedia/Hypertext And Object-Oriented Databases*, pages 17–31. Chapman & Hall, 1991.

[VBD+93] A. Vogel, G.V. Bochmann, R. Dssouli, J. Gecsei, A. Hafid, and B.Kerheve. On QoS negotiation in distributed multimedia applications. Internal report, Université de Montréal, Canada, 1993.

[Vit95] C. Vittal. An object-oriented multimedia database system for a news-on-demand application. Master's thesis, University of Alberta, Department of Computing Science, 1995.

[VÖSEM94] C. Vittal, M.T. Özsu, D. Szafron, and G. El-Medani. The logical design of a multimedia database for a news-on-demand application. Technical report, Department of Computing Science, University of Alberta, 1994.

[Wil91] I. Williams. Hypermedia for multi-user technical documentation. In H. Brown, editor, *Hypermedia/Hypertext And Object-Oriented Databases*, pages 17–31. Chapman & Hall, 1991.

Appendix A

User Profile Class Definitions

User Profile

Purpose

Handles all the settings specified by the user.

Attributes

userName	String
userFilters	Set of filters
userStyleSheets	Set of style sheets
defaultFilter	Reference to the default Filter
defaultStyleSheet	Reference to the default style sheet
defaultMediaSettings	Reference to the default media settings

Methods

get_userName	Returns the user name
get_defaultFilter	Returns a reference to the default filter
get_defaultStyleSheet	Returns a reference to the default style sheet

get_defaultMediaSettings	Returns a reference to the default media settings
set_defaultMediaSettings	Sets the defaultMediaSettings to the given object
add_to_userFilters	Adds the given filter to the set of userFilters
add_to_userStyleSheets	Adds the given style sheet to the userStyleSheets

Filter

Purpose

Handles document filters defined by users

Attributes

name	String
keywords	String
location	String
category	String
source	String
headline	String
authors	String
fromDate	String
toDate	String
checkText	{include, exclude, dontCare}
checkImages	{include, exclude, dontCare}
checkVideo	{include, exclude, dontCare}
checkAudio	{include, exclude, dontCare}

Methods

asString	Returns the values of the filter attributes in the form a string.
get_name	Returns the name of the filter
get_keywords	Returns the keywords string
get_location	Returns the location string
get_category	Returns the category string
get_source	Returns the source string
get_headline	Returns the headline string
get_authors	Returns the authors string
get_fromDate	Returns the fromDate string
get_toDate	Returns the toDate string
get_TextInclude	Returns the checkText value
get_ImagesInclude	Returns the checkImages value
get_AudioInclude	Returns the checkAudio value
get_VideoInclude	Returns the checkVideo value

Style Sheet

Purpose

Contains style sheet information specified by the user. Style sheets are used to display documents.

Attributes

name	String
anchor	{underlined, bordered, iconified, plain}
emphasis1	{italic, bold, underline}
emphasis2	{italic, bold, underline}

font	{times, courier, helvetica}

Methods

asString	Returns the values of the style sheet attributes in the form a string.
get_name	Returns the name of the style sheet
get_anchor	Returns the anchor value
get_emphasis(emphNo)	Returns the emphasis value
get_font	Returns the font value

Media Settings

Purpose

Contains the media settings specified by the user.

Attributes

viewFloatMedia	{immediate, explicit}
linkAudioVideo	{immediate, explicit}
viewTextImage	{inNewWindow, inSameWindow}

Methods

asString	Returns the values of the media settings attributes in the form a string.
get_viewFloatMedia	Returns the viewFloatMedia value
get_linkAudioVideo	Returns the linkAudioVideo value
get_viewTextImage	Returns the viewTextImage value

Appendix B

Valid Command Strings

String commands are sent from the Smalltalk interface to the C++ engine. Any valid command can be divided into three main parts:

- Command Head: contains the name of the command. It must prefixed by **MC_** and must belong to the set of valid commands listed below.
- Name Argument (optional): contains the name of a filter, style sheet or document depending on the particular command. It is prefixed by **MN_**.
- Data Argument (optional): contains information needed by the command. For example, in case of saving a filter, the values of the filter's fields are sent in the data argument. It is prefixed by **MD_**.

Field delimiters are used between the different parts of the command and a *command* delimiter is used at the end of the command string. Thus, the general format of a command is as follows:
MC_*CommandHead*; **MN_***Name*; **MD_***Data*!

Notation:

In the list of commands, italicized words mean that they are a description of a particular string and that the actual string is used in the real implementation.

- MC_getAllDocList!

  Returns a list of document entries in the database, each of which follows the format:

  DocID; Headline

- MC_getCategoryList!

  Returns a list of categories in the database.

- MC_getDefaultFilterName!

  Returns the name of the default filter in the current user profile.

- MC_getDefaultStyleSheetName!

  Returns the name of the default style sheet in the current user profile.

- MC_getDocAbstract; MN_*DocID*!

  Returns the abstract text of the given document.

- MC_getDocAnnotations; MN_*DocID*!

  Returns a list of annotations of the given document. Each annotation is represented by two integers: the starting position and the end position of the annotation object.

  Frontmatter annotation

  Number of section annotations

  Section annotation list

  Number of paragraph annotations

  Paragraph annotation list

  Number of figure annotations

  Figure annotation list

  Number of figure caption annotations

  Figure caption annotation list

  Number of list annotations

  List annotation list

Number of list item annotations

List item annotation list

Number of emphasis1 annotations

Emphasis1 annotation list

Number of emphasis2 annotations

Emphasis2 annotation list

Number of quote annotations

Quote annotation list

Number of link annotations

Link annotation list

- MC_getDocAudioList; MN_*DocID* !

  Returns a list of the names of the audio components of the given document.

- MC_getDocEdinfo; MN_*DocID* !

  Returns a list of the editorial information of the given document:

  Headline

  Location

  Source

  Category

  Date

  Keywords

  Authors

- MC_getDocImage; MN_*DocID*; MD_*ImageName* !

  Returns the size of the image, followed by the binary data of the actual image.

- MC_getDocImageList; MN_*DocID* !

  Returns a list of the names of the images of the given document.

- MC_getDocString; MN_*DocID* !

  Returns the text string of the given document.

- MC_getDocVideoList; MN_*DocID* !

  Returns a list of the names of the video components of the given document.

- MC_getFilterData; MN_*FilterName* !

  Returns the filter (**asString**) with the given name. The filter data follows this format:

  keywords, location, category, source, headline, authors, fromDate, toDate, checkText, checkImages, checkVideo, checkAudio

  The *checkText, checkImages, checkVideo, checkAudio* return one of three strings: include, exclude, dontCare.

- MC_getFilterDocList; MN_*FilterName* !

  Returns a list of document entries filtered on the filter with the given name. Entries follow this format:

  DocID; Headline

- MC_getFilterList!

  Returns a list of filter names in the current user profile.

- MC_getLocationList!

  Returns a list of locations in the database.

- MC_getMediaData!

  Returns the media settings (**asString**) in the current user profile in the following format:

  viewFloatMedia, linkAudioVideo, viewTextImage

  The first two fields return one of two strings: immediate or explicit. The last field returns either newWindow or sameWindow.

- MC_getSearchResult; MN_*FilterName-OR-DocID*; MD_*queryString, searchText, searchAllText, searchHeadlines, searchAbstracts, searchSections, searchParagraphs, searchLists, searchQuotes, searchEmphasis, searchAuthors, searchCategory, searchKeywords, searchLocation, searchSource, searchImages, searchAudio, searchVideo,*

searchScope, retrieveDocuments, retrieveComponents !

The *FilterName-OR-DocID* is used to determine the scope of the search. In case of searching all the database, this field is ignored. All the search and retrieve fields are either *true* or *false* depending on whether the user wishes to search/retrieve these object types or not.

Returns a list of objects which match the query string. The type of the objects returned is dependent on the search and retrieve fields. If the user is searching all the media types and retrieving all the components, the following is returned:

Number of Articles

'DocID;Headline' list

Number of Headlines

Headlines list

Number of Abstracts

'DocID; Headline, Abstract summary' list

Number of Sections

'DocID; Headline, Section summary' list

Number of Paragraphs

'DocID; Headline, Paragraph summary' list

Number of Quotes

'DocID; QuoteName; Headline, Quote summary' list

Number of Emphasis

'DocID; Headline, Emphasis summary' list

Number of Authors

Authors list

Number of Categories

Category list

Number of Keywords

Keyword list

Number of locations

Location list

Number of Source

Source list

Number of Images

'DocID; ImageName; Headline, ImageName, Summary' list

Number of Audio

'DocID; AudioName; Headline, AudioName, Summary' list

Number of Video

'DocID; VideoName; Headline, VideoName, Summary' list

- MC_getSourceList!

  Returns a list of media sources in the database.

- MC_getStyleSheetData; MN_*StyleSheetName* !

  Returns the style sheet (**asString**) with the given name. The data follows this format:

  anchor, emphasis1, emphasis2, font

  The *anchor* string can be: underlined, bordered, iconified, or plain. The *emphasis1* and *emphasis2* string can be: italic, bold, or underline. The *font* string can be: times, courier, helvetica.

- MC_getStyleSheetList!

  Returns a list of style sheet names in the current user profile.

- MC_setFilterData; MN_*FilterName*; MD_*keywords, location, category, source, headline, authors, fromDate, toDate, checkText, checkImages, checkVideo, checkAudio* !

  Returns a positive acknowledgment or an error.

- MC_setMediaData; MD_*viewFloatMedia, linkAudioVideo, viewTextImage* !

  Returns a positive acknowledgment or an error.

- MC_setStyleSheetData; MN_*StyleSheetName*; MD_*anchor*, *emphasis1*, *emphasis2*, *font* !

  Returns a positive acknowledgment or an error.

- MC_quitProgram!

  Returns a string indicating that the program is terminated.

Appendix C

Glossary

Abstraction	The ability to encapsulate and isolate design and execution information of an object, providing only a set of operations to interface with this object [Bud91].
Annotation	A pair of integers defining the start and the end position of a text element with respect to its document's entire text string. Annotations are used to avoid fragmenting the text with each element.
Attribute	A qualifier indicating a property of an element other than its type and its content [Her94]. For instance, the date of publication is an attribute of a news article.
Browsing	Looking through material in a casual manner; usually to locate information of interest for further investigation.
Continuous Media	Media types with real-time constraints such as audio and video. Real-time constraints are needed for synchronization and playback.
Customization	Allowing users to define their own settings of the system according to their prefernce.
Document	A collection of related information nodes containing any combination of media types: text, images, audio or video. A multimedia news article is a document according to this definition.

DTD	Document type definition. The definition of markup rules and constraints defined for a given class of SGML documents [Her94].
Filter	To specify the scope of documents a user views, depending on certain criteria such as keyword(s), location, headline, etc. This is done by defining filter objects and applying them to the database. The result is a set of documents whose attributes match the ones specified by the user in the filter. In other words, filtering is a search on document attributes, rather than content.
Inheritance	The property of objects by which instances of a class can have access to data and method definitions contained in a previously defined class, without those definitions being restated [Bud91].
Hypermedia	Generalization of hypertext (see below) where nodes may contain any media type: text, images, audio, and video.
Hypertext	A non-sequential organization of text which presents several options for readers to follow. Pieces of information (nodes) are linked to other nodes containing related information.
HyTime	The International Standard for Hypermedia and Time based systems, ISO 10744:1992.
Media Settings	Store presentation settings concerned with the display of the various media types in the system.
Monomedia	Objects containing only one media type. For example, text objects or audio streams.
Navigation	Traversing through information by following hypertext links from one node to another.
News Providers	Agencies responsible for producing multimedia news articles and inserting them in the database. Examples od news providers are television networks, newspapers, magazines, and wire services.

Non-Continuous Media	Static media types that do not have any real-time requirements, such as text and images.
Pipe	A connection between two processes that takes the output of one process and pump it as input to the other.
Quality-of-service management	Ensures that a satisfactory quality-of-service is provided to users of multimedia, especially with real-time delivery and quality of continuous media.
Query	A request that is performed on the database to retrieve specific information matching a certain selection criterion.
Search	Allows users to search documents' content for specific information. The search criteria is defined by a search string, the scope of the search, the media types to be searched, and the returned types.
SGML	The Standard Generalized Markup Language, ISO 8879. Defined by the standard as "A language for document representation that formalizes markup and frees it of system and processing dependencies." An abstract language, with which an arbitrary number of markup languages may be defined [Her94].
Style Sheet	Stores user preferences with regards to presenation of documents. This includes font formats, presentation of anchors, indentation of paragraphs, etc.
Type System	A hierarchy of types corresponding to the logical elements in the system. The type system defines the types of objects and the relationships between them needed to model the target application.
User Profile	Contains all user preferences. User profiles are used for system customization. They contain filters, style sheets, media settings, and system defaults.